	Type	L #	Hits	Search Text	DBs	Time Stamp
1	IS&R	L1	1	("5809251").PN.	USPAT; US-PGPUB	2002/12/05 13:49
2	IS&R	L2	1	("5860012").PN.	USPAT; US-PGPUB	2002/12/05 14:02
3	BRS	L4	250	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (successful or unsuccessful or retry)	USPAT; US-PGPUB	2002/12/05 14:06
4	BRS	L5	8	L4 and 717/$.ccls.	USPAT; US-PGPUB	2002/12/05 14:04
5	BRS	L6	1	L5 and queue	USPAT; US-PGPUB	2002/12/05 14:05
6	BRS	L7	366	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (successful$3 or unsuccessful$3 or retry$1)	USPAT; US-PGPUB	2002/12/05 14:11
7	BRS	L9	3	L8 and queu$4	USPAT; US-PGPUB	2002/12/05 14:06
8	BRS	L8	17	L7 and 717/$.ccls.	USPAT; US-PGPUB	2002/12/05 14:07
9	BRS	L10	2	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (retry$1 or reexecut$4 or re-execut$4) and 717/$.ccls.	USPAT; US-PGPUB	2002/12/05 14:20
10	BRS	L11	8	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (retr$4 or reexecut$4 or re-execut$4) and 717/$.ccls.	USPAT; US-PGPUB	2002/12/05 14:17

	Type	L #	Hits	Search Text	DBs	Time Stamp
11	BRS	L12	3	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (retry$1 or reexecut$4 or re-execut$4 or rerun$4 or re-run$4) and 717/$.ccls.	USPAT; US-PGPUB	2002/12/05 14:37
12	BRS	L13	100	(verif$7 or validat$5) near5 (update or execut$5 or install5) same (retry$1 or reexecut$4 or re-execut$4 or rerun$4 or re-run$4)	USPAT; US-PGPUB	2002/12/05 14:37

US005809251A

United States Patent [19]

May et al.

[11] Patent Number: 5,809,251

[45] Date of Patent: Sep. 15, 1998

[54] **REMOTE INSTALLATION OF SOFTWARE BY A MANAGEMENT INFORMATION SYSTEM INTO A REMOTE COMPUTER**

[75] Inventors: **Gregory J. May**, Corvallis, Oreg.; **Colin I'Anson**, Bristol, England

[73] Assignee: **Hewlett-Packard Company**, Palo Alto, Calif.

[21] Appl. No.: **728,004**

[22] Filed: **Oct. 9, 1996**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/200.53**; 395/200.69; 395/712

[58] **Field of Search** 395/712, 619, 395/651, 200.01, 200.09, 200.12, 200.14, 200.02, 200.3, 200.47, 200.57, 200.67, 200.48, 200.68, 200.69, 200.55, 200.53, 200.56; 707/203

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,636,950	1/1987	Caswell et al.	395/228
4,788,637	11/1988	Tamaru	395/200.1
5,008,814	4/1991	Mathur	395/200.09
5,043,721	8/1991	May	340/825.44
5,155,847	10/1992	Kirouac et al.	395/200.1
5,293,484	3/1994	Dabbs, III et al.	395/201
5,337,044	8/1994	Folger et al.	340/825.44
5,367,452	11/1994	Gallery et al.	395/228
5,387,904	2/1995	Takada	340/825.44
5,406,643	4/1995	Burke et al.	395/200.12
5,444,438	8/1995	Goldberg	340/825.44
5,577,244	11/1996	Killebrew et al.	395/703
5,581,703	12/1996	Baugher et al.	395/200.55

OTHER PUBLICATIONS

"IBM TME 10 NetFinity Server Up and Running", Jun. 1996, IBM, US XP002055592, Publication No. SH19-4302-00, p. 120, line 35–p. 124, line 4; figure 68.

"Electronic Phone Book For Video Conferencing", IBM Technical Disclosure Bulletin, vol. 36, No. 6A, 1 Jun. 1993, pp. 57–61, XP000370759, the whole document.

Database IAC Newsletter Collection, Information Access Company, XP002055593, & "Epilogue Technology, Xircom finalize mobile MIB proposal", The Osinetter Newsletter, vol. 10 No. 6, Jun. 1995, the whole document.

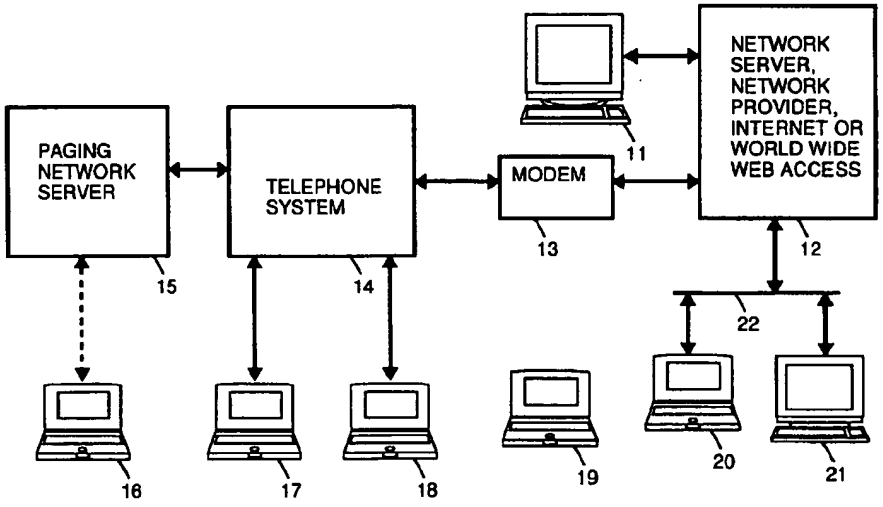
"Inter–Process Communications Library", IBM Technical Disclosure Bulletin, vol. 36, No. 1, 1 Jan. 1993, pp. 59–60, XP00033372.

Primary Examiner—Parshotam S. Lall
Assistant Examiner—Viet Vu

[57] **ABSTRACT**

Remote installation of an update of software is forwarded by a management information system into a remote computer. When a primary communication path between the management information system and the remote computer is unavailable and an alternate communication path is available, a connection is established between the management information system and the remote computer using the alternate communication path. The management information system requests from the remote computer the current version information about the software within the remote computer. When the management information system determines the current version of the software within the remote computer needs to be updated, the management information system determines whether the alternate communication path is adequate for downloading the update of the software. When the management information system determines that the alternate communication path is adequate for downloading the update of the software, the update of the software is downloaded from the management information system to the remote computer.

22 Claims, 18 Drawing Sheets



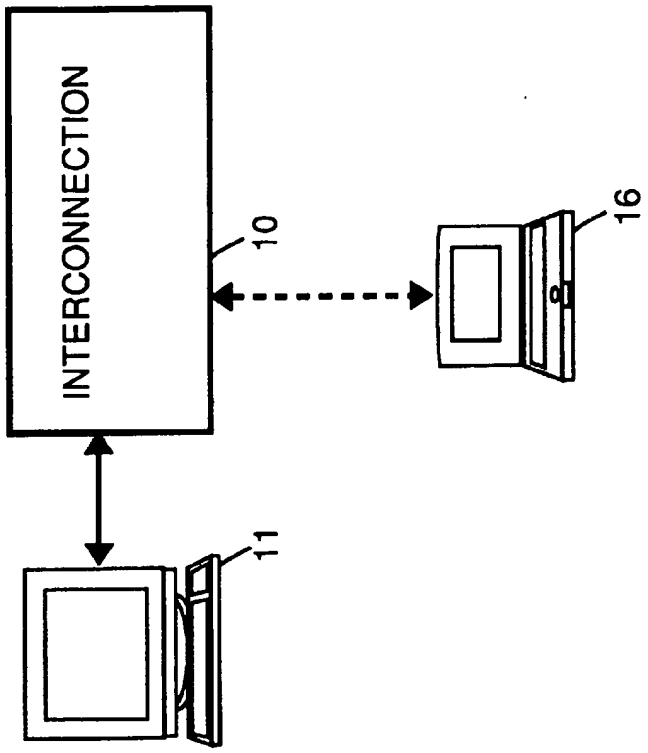


FIGURE 1

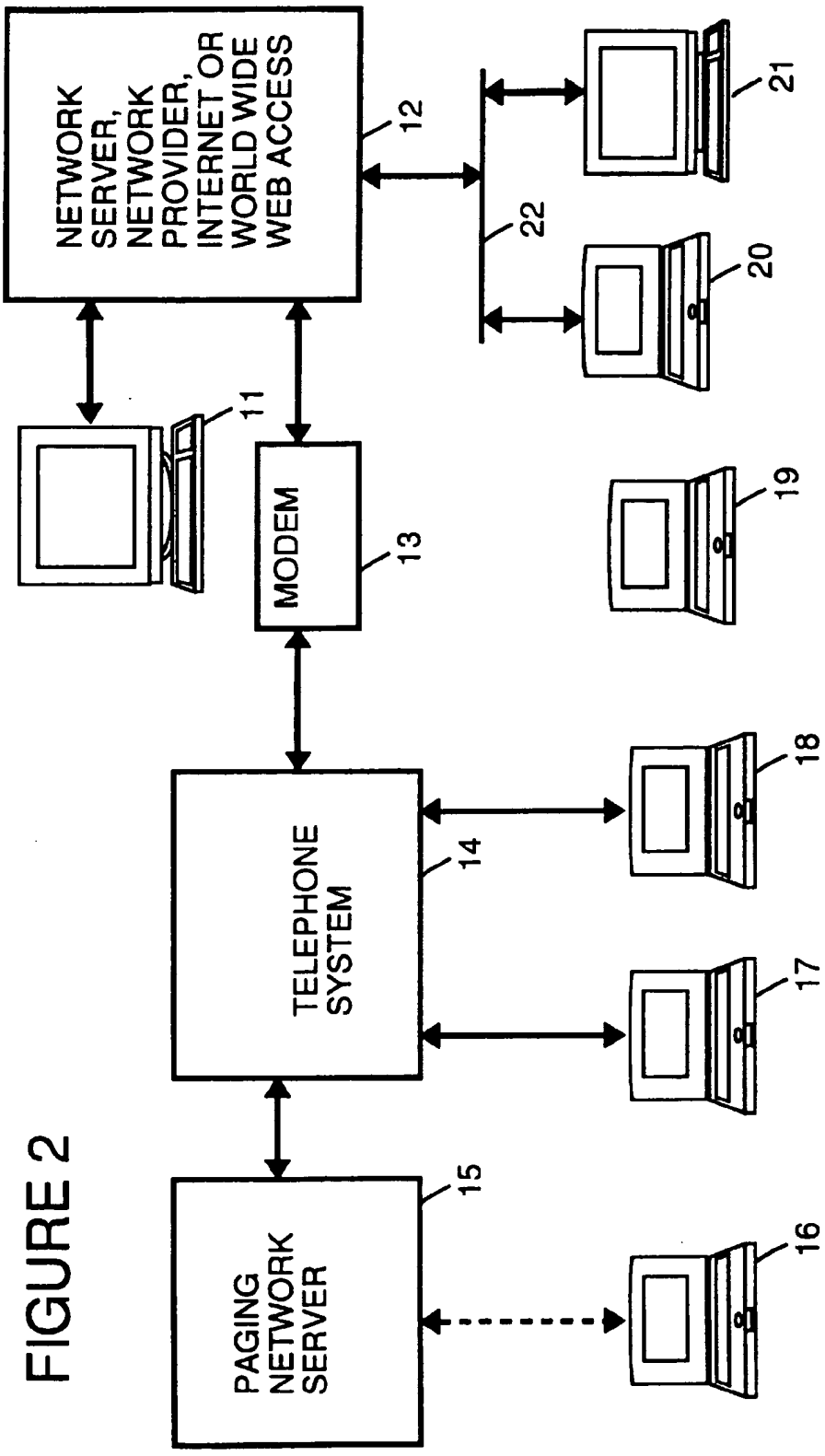

NETWORK SERVER, NETWORK PROVIDER, INTERNET OR WORLD WIDE WEB ACCESS
12
21
22
20
11
MODEM
13
19
TELEPHONE SYSTEM
14
18
17
PAGING NETWORK SERVER
15
16

FIGURE 2

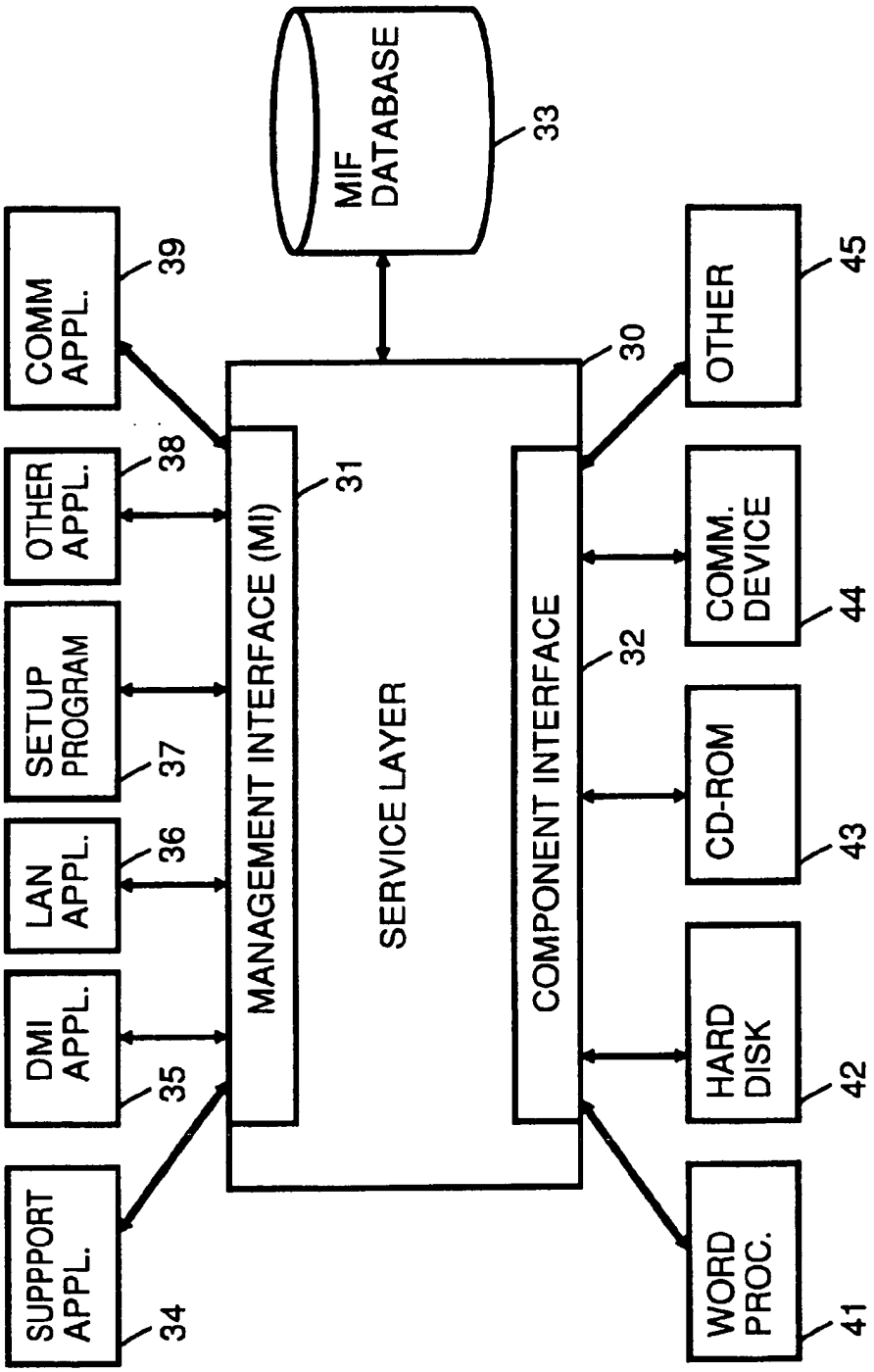

MIF DATABASE
33
COMM APPL.
39
OTHER APPL.
38
SETUP PROGRAM
37
LAN APPL.
36
DMI APPL.
35
SUPPPORT APPL.
34
MANAGEMENT INTERFACE (MI)
31
SERVICE LAYER
COMPONENT INTERFACE
30
32
OTHER
45
COMM. DEVICE
44
CD-ROM
43
HARD DISK
42
WORD PROC.
41

FIGURE 3

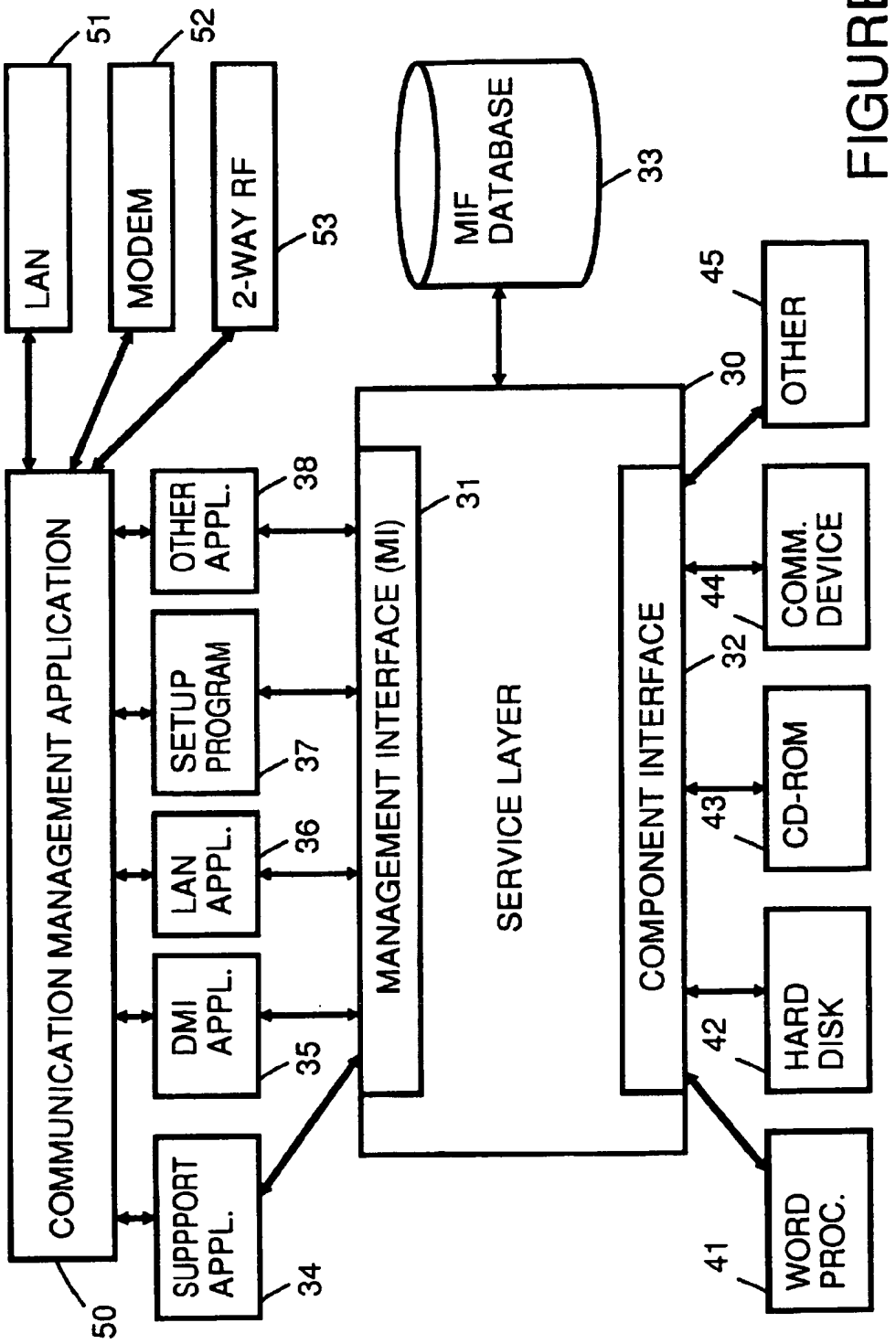
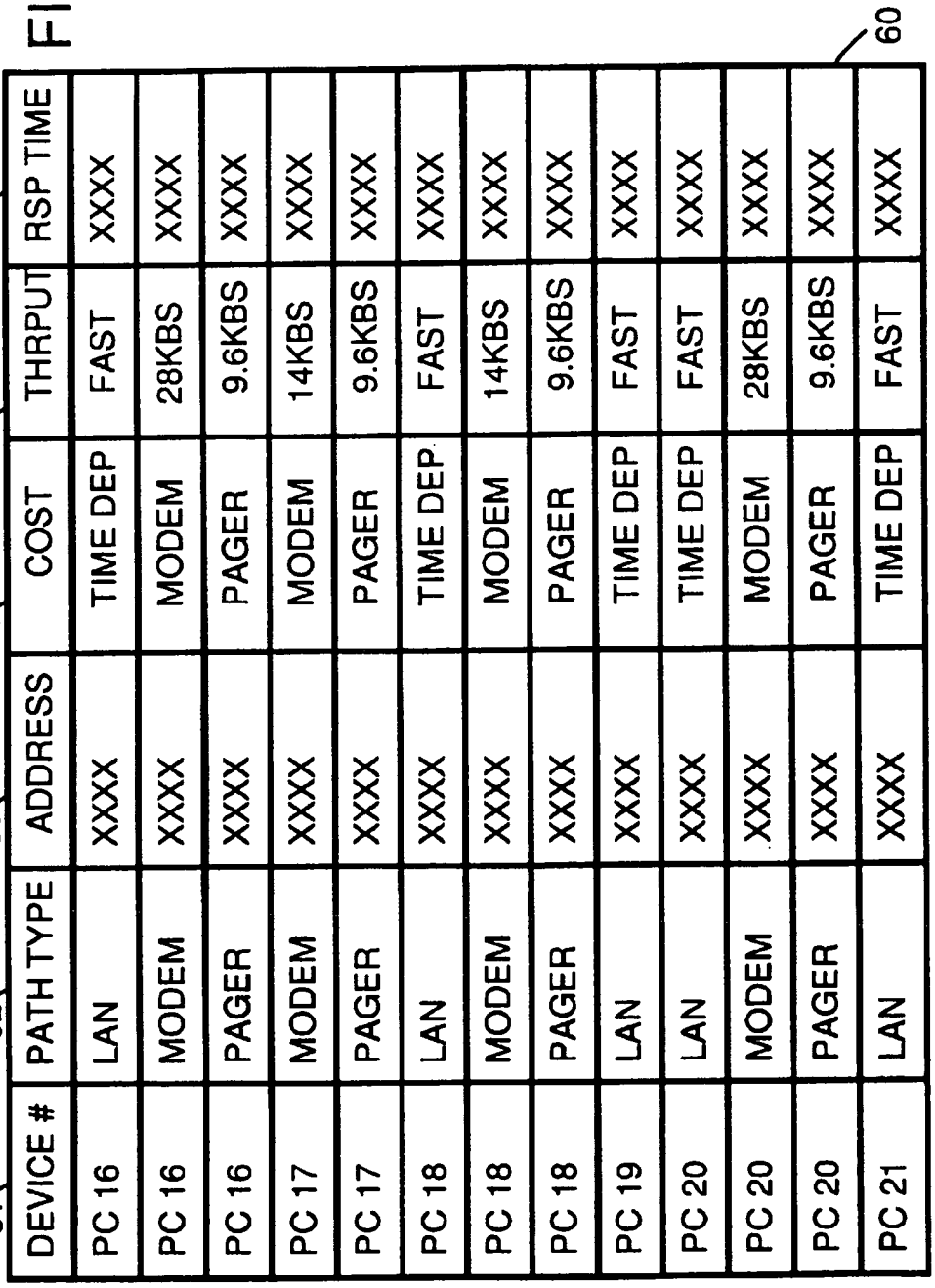

LAN
MODEM
2-WAY RF
51
52
53
MIF DATABASE
33
COMMUNICATION MANAGEMENT APPLICATION
50
OTHER APPL.
38
SETUP PROGRAM
37
LAN APPL.
36
DMI APPL.
35
SUPPPORT APPL.
34
MANAGEMENT INTERFACE (MI)
31
SERVICE LAYER
COMPONENT INTERFACE
30
OTHER
45
COMM. DEVICE
44
32
CD-ROM
43
HARD DISK
42
WORD PROC.
41

FIGURE 4

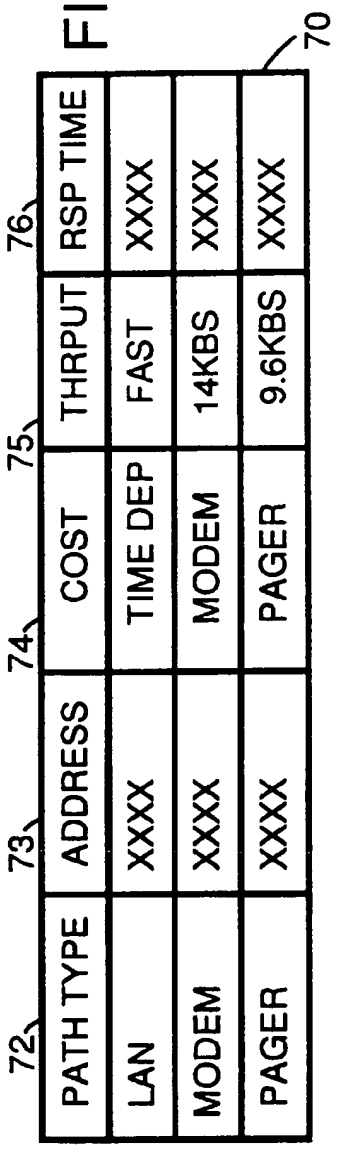
FIGURE 5



61 DEVICE #	62 PATH TYPE	63 ADDRESS	64 COST	65 THRPUT	66 RSP TIME
PC 16	LAN	XXXX	TIME DEP	FAST	XXXX
PC 16	MODEM	XXXX	MODEM	28KBS	XXXX
PC 16	PAGER	XXXX	PAGER	9.6KBS	XXXX
PC 17	MODEM	XXXX	MODEM	14KBS	XXXX
PC 17	PAGER	XXXX	PAGER	9.6KBS	XXXX
PC 18	LAN	XXXX	TIME DEP	FAST	XXXX
PC 18	MODEM	XXXX	MODEM	14KBS	XXXX
PC 18	PAGER	XXXX	PAGER	9.6KBS	XXXX
PC 19	LAN	XXXX	TIME DEP	FAST	XXXX
PC 20	LAN	XXXX	TIME DEP	FAST	XXXX
PC 20	MODEM	XXXX	MODEM	28KBS	XXXX
PC 20	PAGER	XXXX	PAGER	9.6KBS	XXXX
PC 21	LAN	XXXX	TIME DEP	FAST	XXXX

60

FIGURE 6


70
72
73
74
75
76
PATH TYPE
ADDRESS
COST
THRPUT
RSP TIME
LAN
XXXX
TIME DEP
FAST
XXXX
MODEM
XXXX
MODEM
14KBS
XXXX
PAGER
XXXX
PAGER
9.6KBS
XXXX

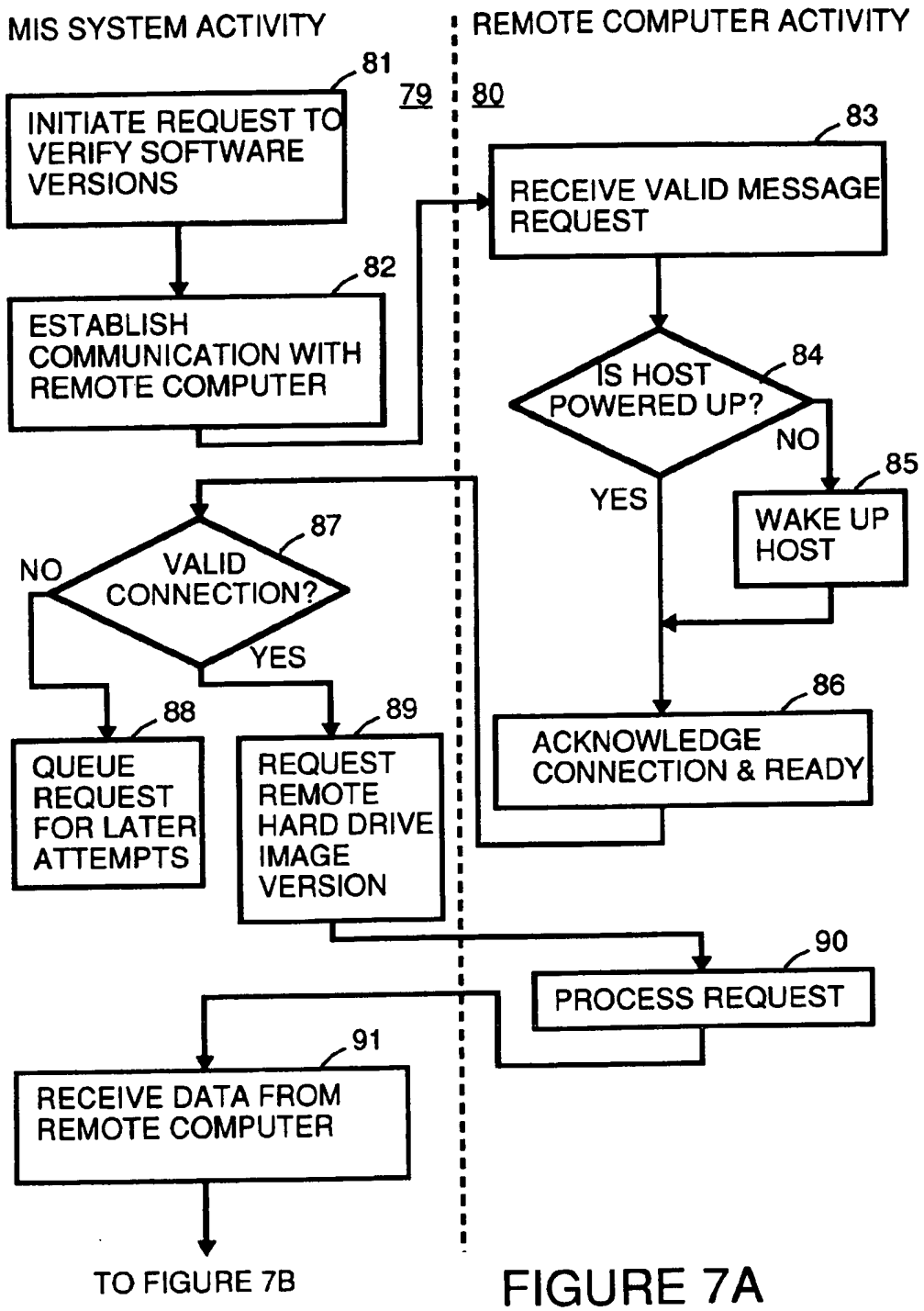

MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
79
80
81
INITIATE REQUEST TO VERIFY SOFTWARE VERSIONS
82
ESTABLISH COMMUNICATION WITH REMOTE COMPUTER
83
RECEIVE VALID MESSAGE REQUEST
84
IS HOST POWERED UP?
NO
YES
85
WAKE UP HOST
87
VALID CONNECTION?
NO
YES
86
ACKNOWLEDGE CONNECTION & READY
88
QUEUE REQUEST FOR LATER ATTEMPTS
89
REQUEST REMOTE HARD DRIVE IMAGE VERSION
90
PROCESS REQUEST
91
RECEIVE DATA FROM REMOTE COMPUTER
TO FIGURE 7B

FIGURE 7A

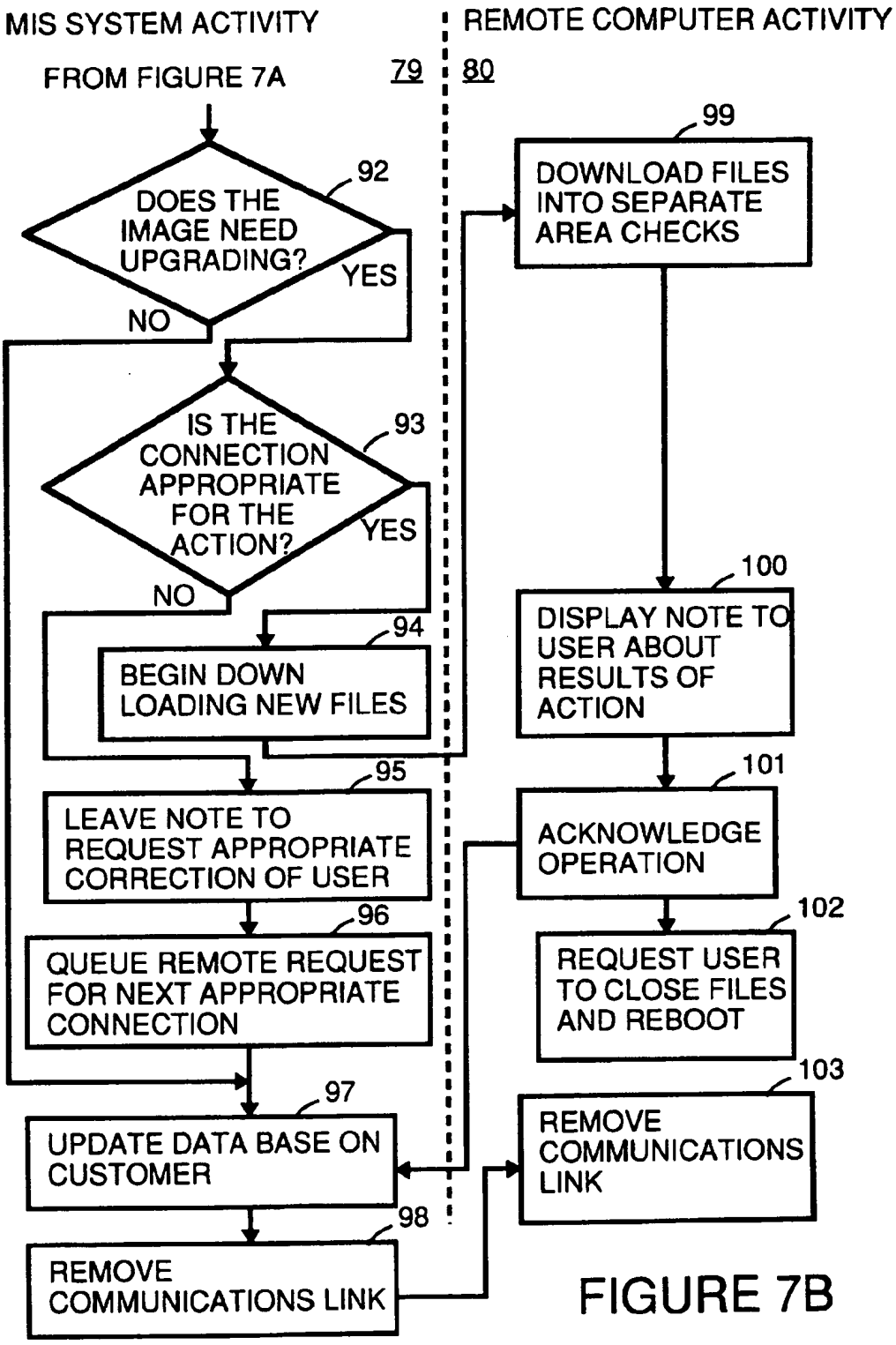

MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
FROM FIGURE 7A
79
80
DOES THE IMAGE NEED UPGRADING?
92
YES
NO
IS THE CONNECTION APPROPRIATE FOR THE ACTION?
93
YES
NO
BEGIN DOWN LOADING NEW FILES
94
LEAVE NOTE TO REQUEST APPROPRIATE CORRECTION OF USER
95
QUEUE REMOTE REQUEST FOR NEXT APPROPRIATE CONNECTION
96
UPDATE DATA BASE ON CUSTOMER
97
REMOVE COMMUNICATIONS LINK
98
DOWNLOAD FILES INTO SEPARATE AREA CHECKS
99
DISPLAY NOTE TO USER ABOUT RESULTS OF ACTION
100
ACKNOWLEDGE OPERATION
101
REQUEST USER TO CLOSE FILES AND REBOOT
102
REMOVE COMMUNICATIONS LINK
103

FIGURE 7B

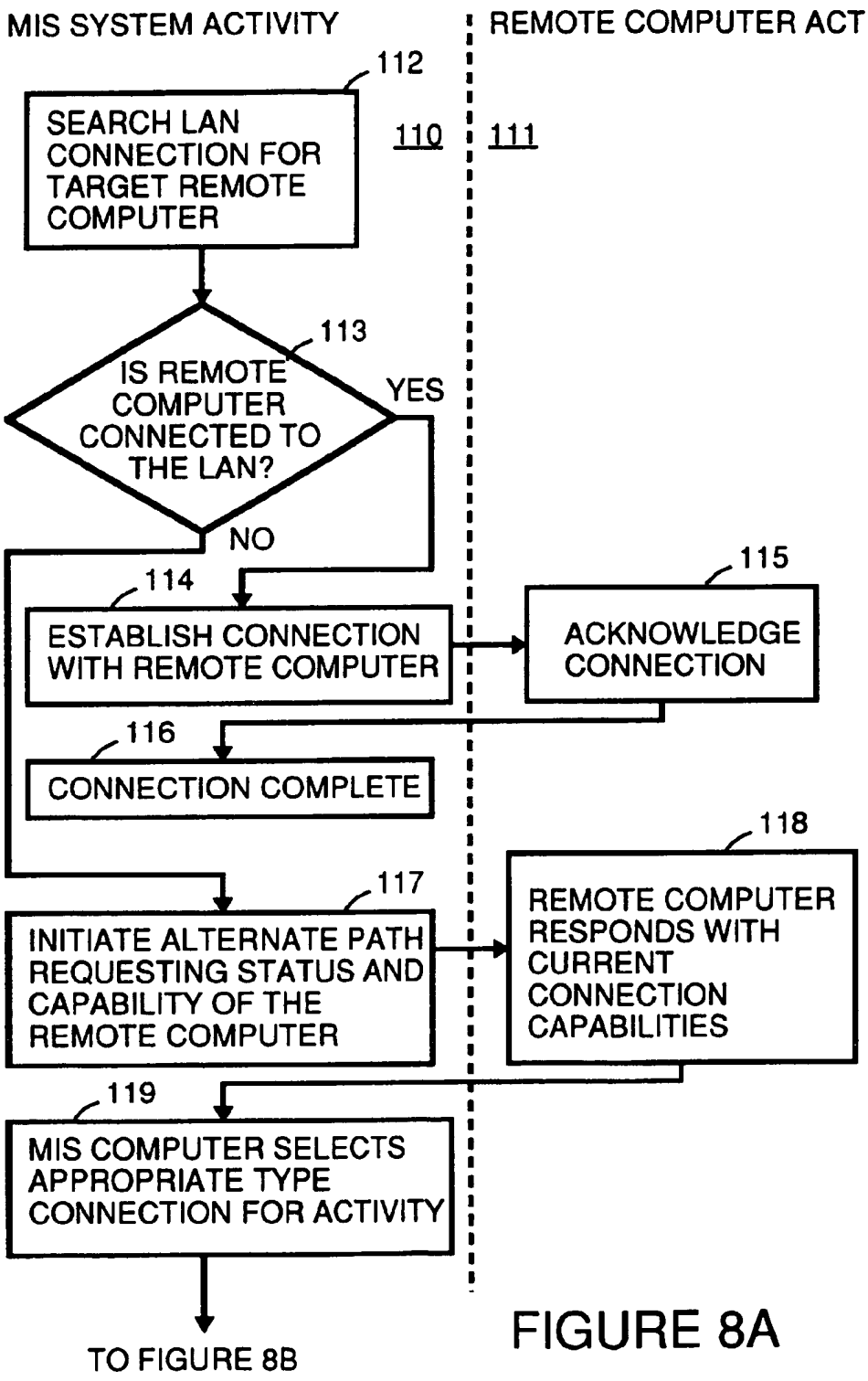

MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
112
SEARCH LAN CONNECTION FOR TARGET REMOTE COMPUTER
110
111
113
IS REMOTE COMPUTER CONNECTED TO THE LAN?
YES
NO
114
ESTABLISH CONNECTION WITH REMOTE COMPUTER
115
ACKNOWLEDGE CONNECTION
116
CONNECTION COMPLETE
118
117
INITIATE ALTERNATE PATH REQUESTING STATUS AND CAPABILITY OF THE REMOTE COMPUTER
REMOTE COMPUTER RESPONDS WITH CURRENT CONNECTION CAPABILITIES
119
MIS COMPUTER SELECTS APPROPRIATE TYPE CONNECTION FOR ACTIVITY
TO FIGURE 8B

FIGURE 8A

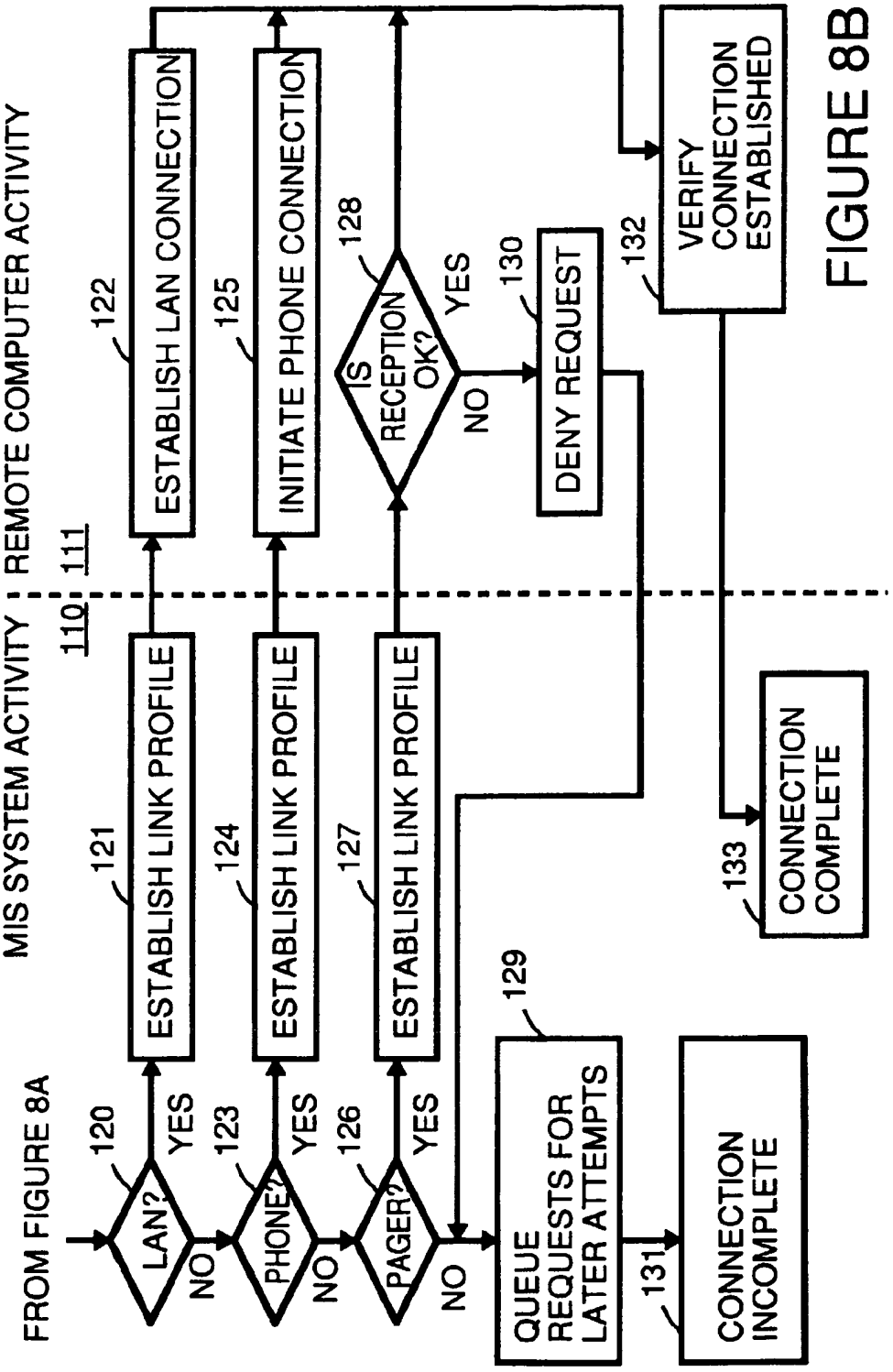

FROM FIGURE 8A
MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
110
111
LAN?
120
YES
NO
ESTABLISH LINK PROFILE
121
ESTABLISH LAN CONNECTION
122
PHONE?
123
YES
NO
ESTABLISH LINK PROFILE
124
INITIATE PHONE CONNECTION
125
PAGER?
126
YES
NO
ESTABLISH LINK PROFILE
127
IS RECEPTION OK?
128
YES
NO
DENY REQUEST
130
QUEUE REQUESTS FOR LATER ATTEMPTS
129
CONNECTION INCOMPLETE
131
VERIFY CONNECTION ESTABLISHED
132
CONNECTION COMPLETE
133

FIGURE 8B

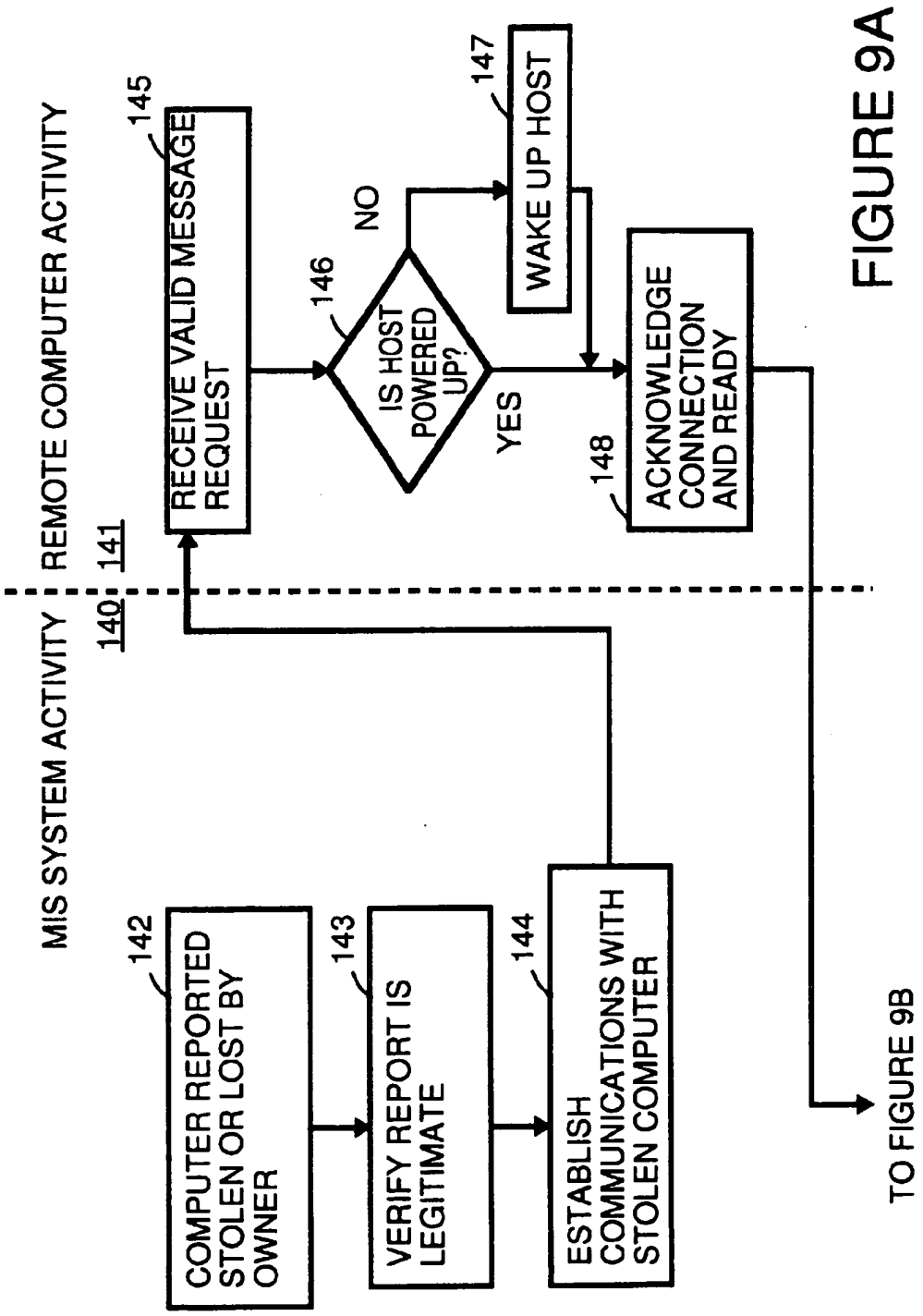

MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
140
141
142
COMPUTER REPORTED STOLEN OR LOST BY OWNER
143
VERIFY REPORT IS LEGITIMATE
144
ESTABLISH COMMUNICATIONS WITH STOLEN COMPUTER
145
RECEIVE VALID MESSAGE REQUEST
146
IS HOST POWERED UP?
NO
YES
147
WAKE UP HOST
148
ACKNOWLEDGE CONNECTION AND READY
TO FIGURE 9B

FIGURE 9A

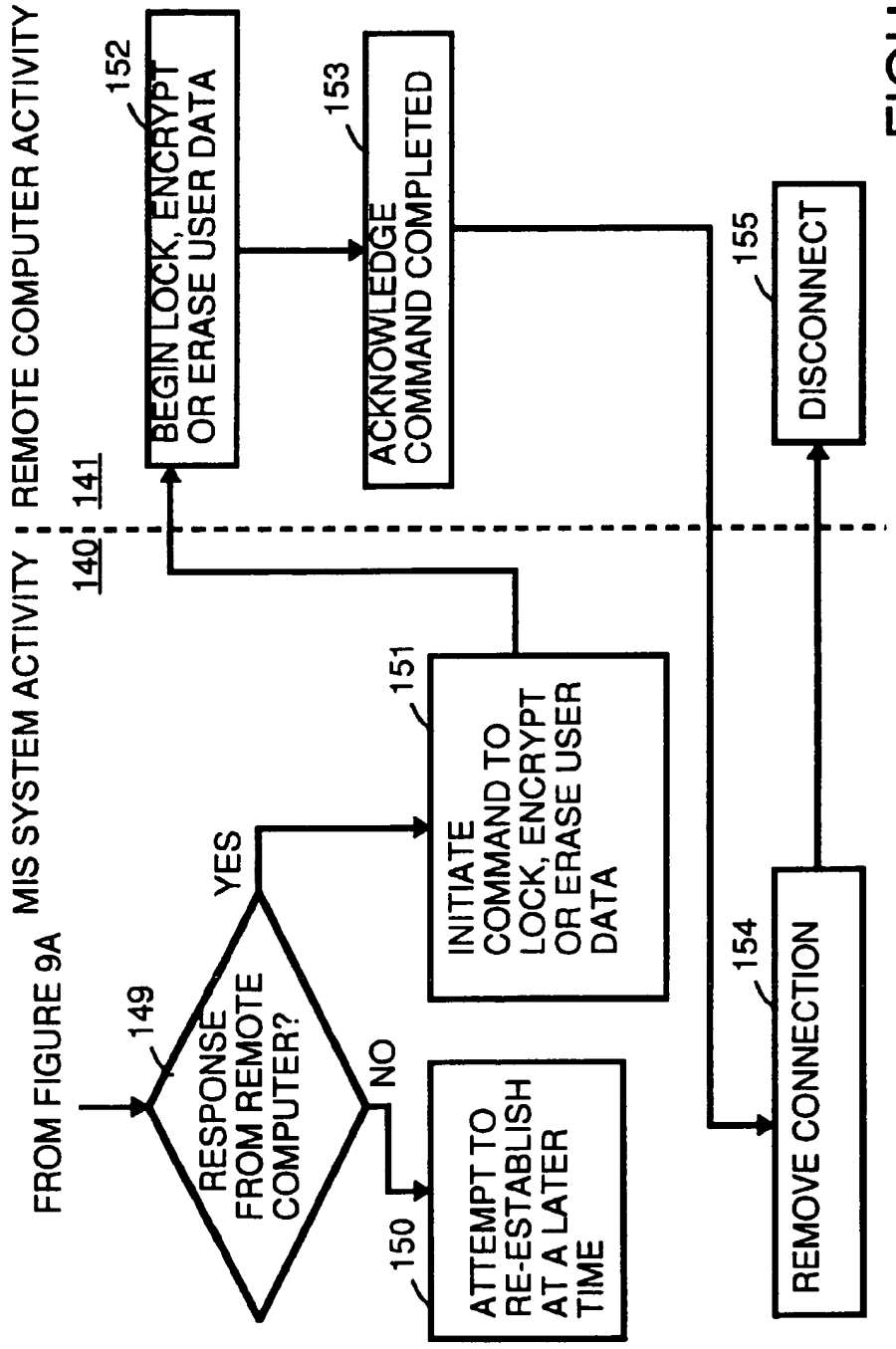

FROM FIGURE 9A
MIS SYSTEM ACTIVITY
REMOTE COMPUTER ACTIVITY
140
141
149
RESPONSE FROM REMOTE COMPUTER?
YES
NO
150
ATTEMPT TO RE-ESTABLISH AT A LATER TIME
151
INITIATE COMMAND TO LOCK, ENCRYPT OR ERASE USER DATA
152
BEGIN LOCK, ENCRYPT OR ERASE USER DATA
153
ACKNOWLEDGE COMMAND COMPLETED
154
REMOVE CONNECTION
155
DISCONNECT

FIGURE 9B

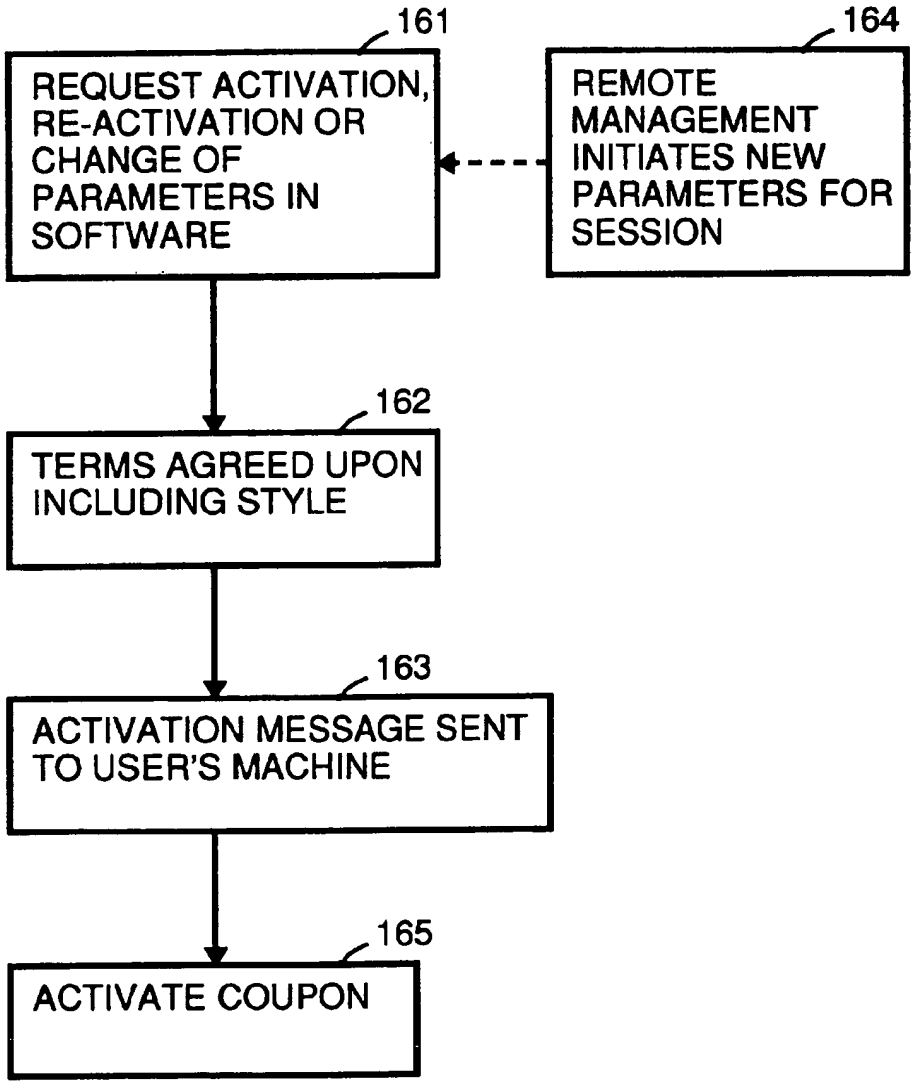

161
REQUEST ACTIVATION, RE-ACTIVATION OR CHANGE OF PARAMETERS IN SOFTWARE
164
REMOTE MANAGEMENT INITIATES NEW PARAMETERS FOR SESSION
162
TERMS AGREED UPON INCLUDING STYLE
163
ACTIVATION MESSAGE SENT TO USER'S MACHINE
165
ACTIVATE COUPON

FIGURE 10

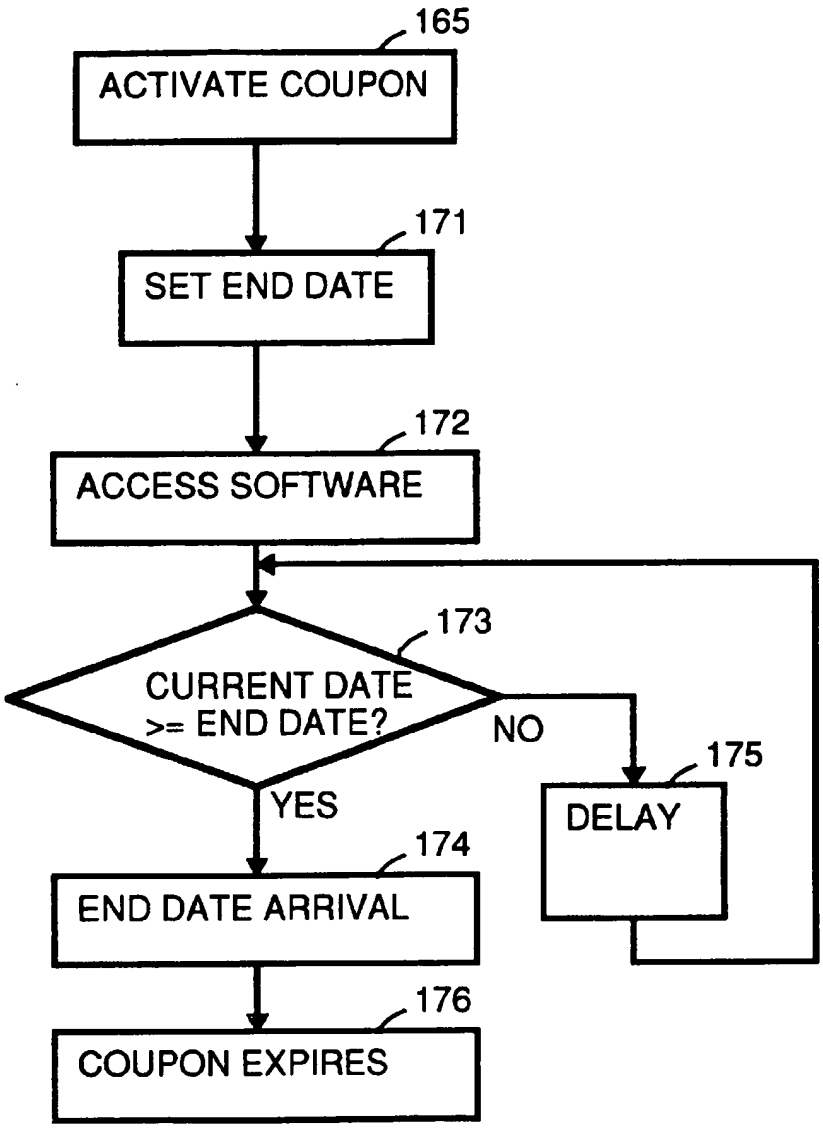

165
ACTIVATE COUPON
171
SET END DATE
172
ACCESS SOFTWARE
173
CURRENT DATE
>= END DATE?
NO
YES
175
DELAY
174
END DATE ARRIVAL
176
COUPON EXPIRES

FIGURE 11

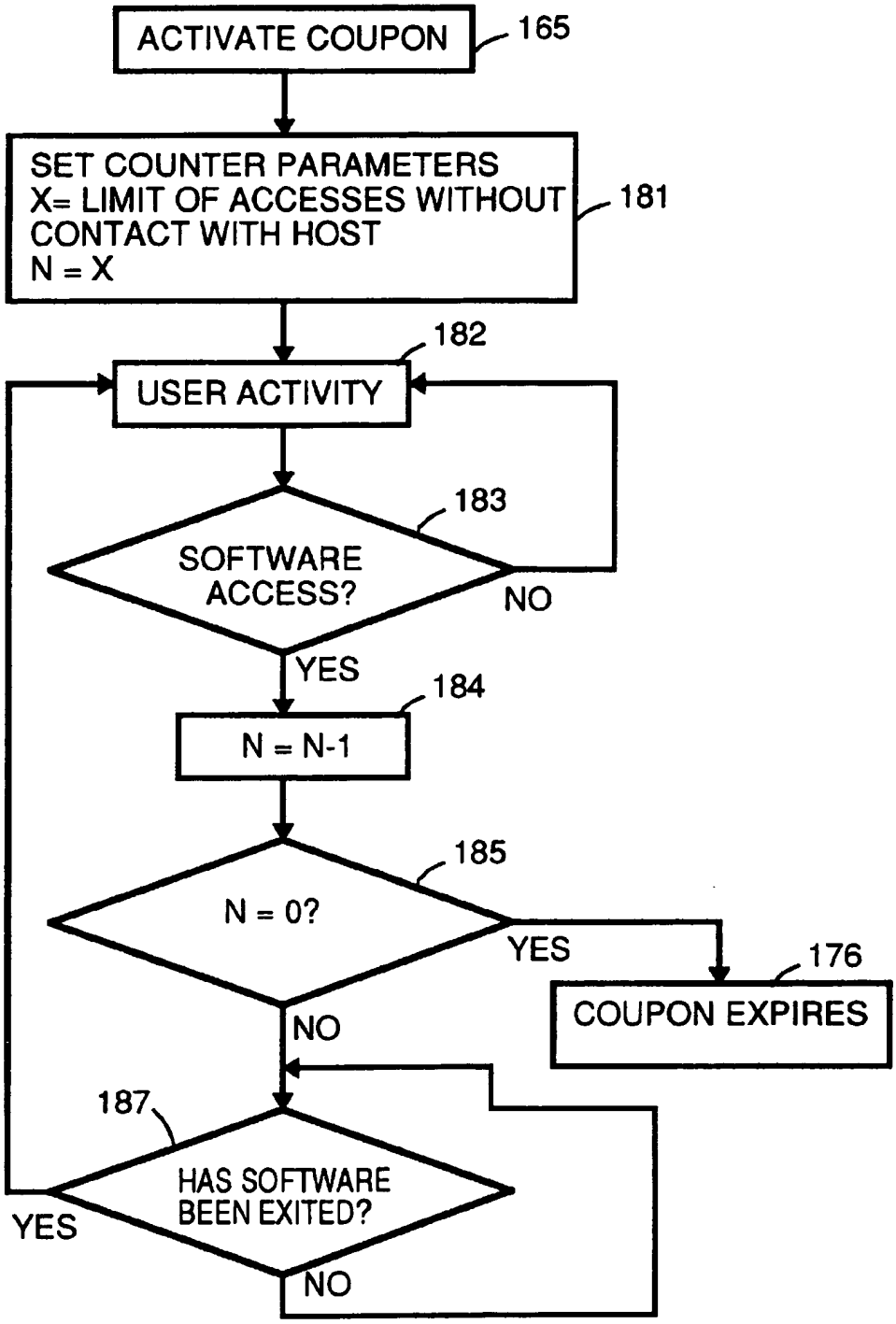

ACTIVATE COUPON
165
SET COUNTER PARAMETERS
X= LIMIT OF ACCESSES WITHOUT
CONTACT WITH HOST
N = X
181
182
USER ACTIVITY
183
SOFTWARE
ACCESS?
NO
YES
184
N = N-1
185
N = 0?
YES
176
COUPON EXPIRES
NO
187
HAS SOFTWARE
BEEN EXITED?
YES
NO

FIGURE 12

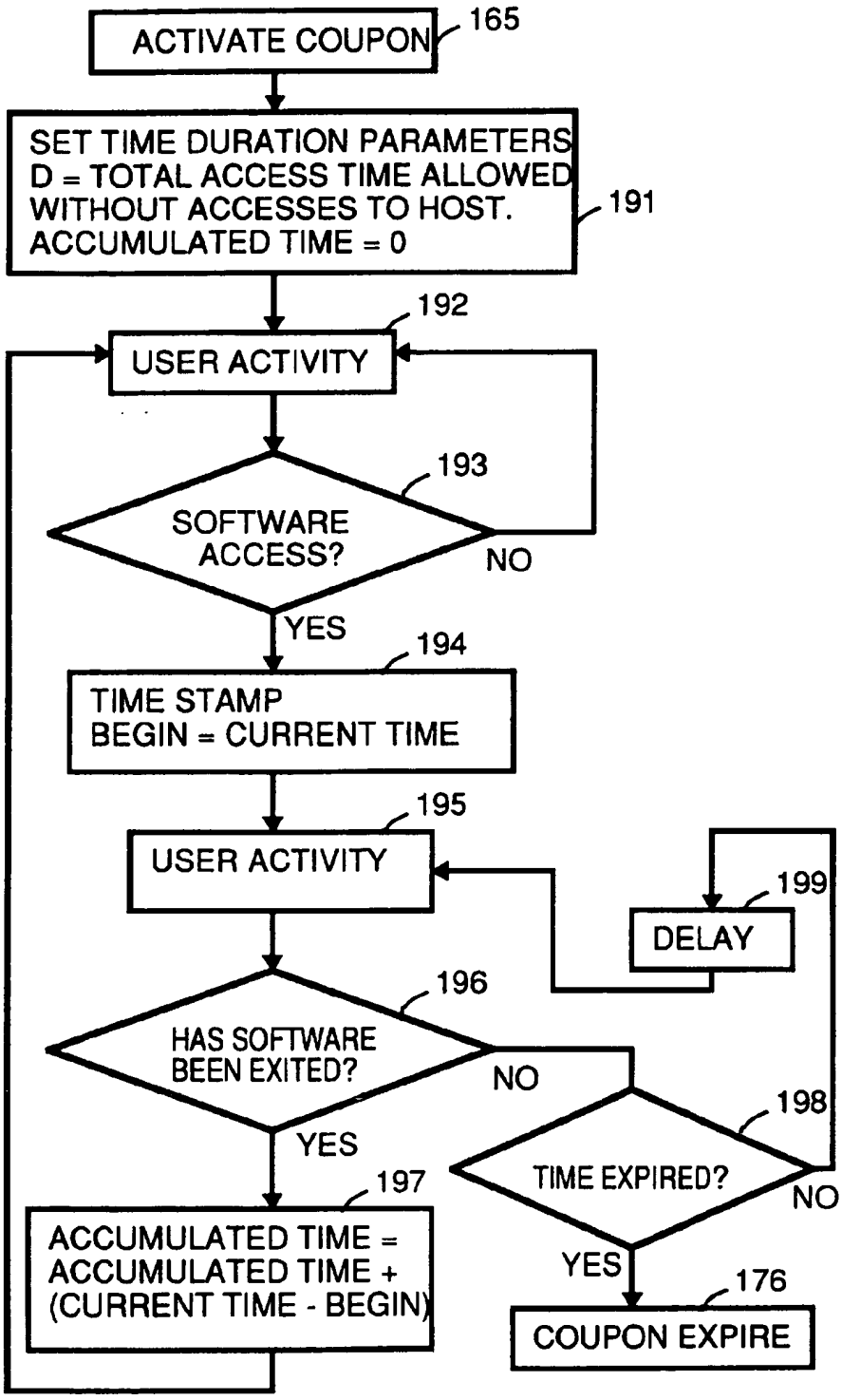

ACTIVATE COUPON
165
SET TIME DURATION PARAMETERS
D = TOTAL ACCESS TIME ALLOWED
WITHOUT ACCESSES TO HOST.
ACCUMULATED TIME = 0
191
192
USER ACTIVITY
193
SOFTWARE ACCESS?
NO
YES
194
TIME STAMP
BEGIN = CURRENT TIME
195
USER ACTIVITY
199
DELAY
196
HAS SOFTWARE BEEN EXITED?
NO
YES
198
TIME EXPIRED?
NO
YES
197
ACCUMULATED TIME = ACCUMULATED TIME + (CURRENT TIME - BEGIN)
176
COUPON EXPIRE

FIGURE 13

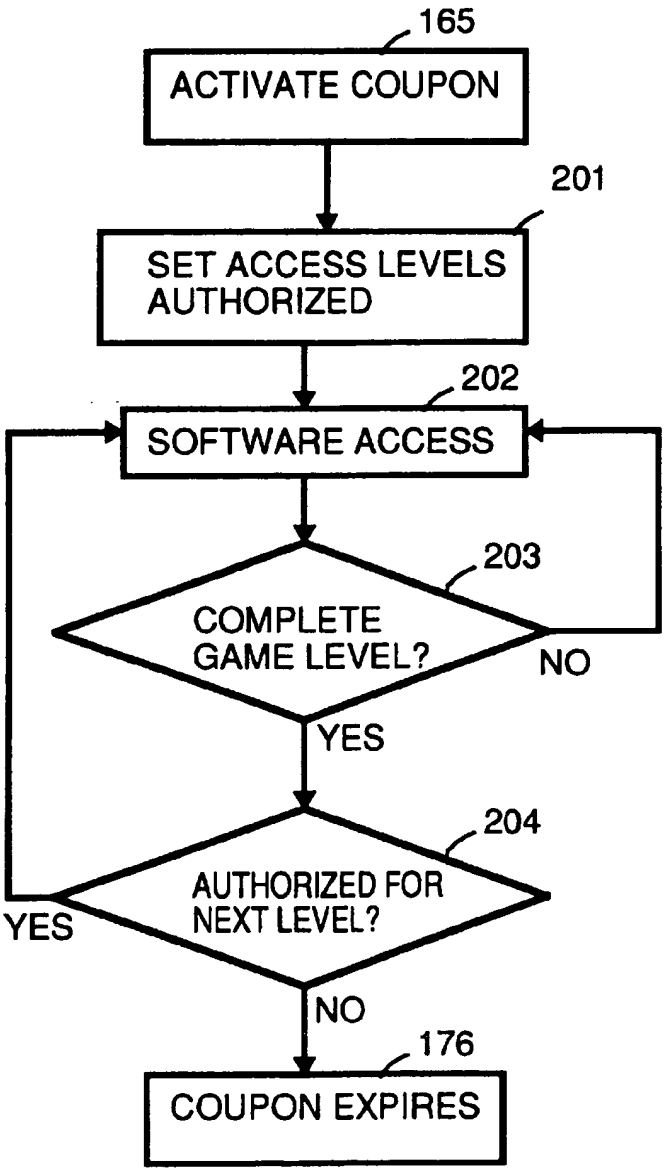

165
ACTIVATE COUPON
201
SET ACCESS LEVELS AUTHORIZED
202
SOFTWARE ACCESS
203
COMPLETE GAME LEVEL?
NO
YES
204
AUTHORIZED FOR NEXT LEVEL?
YES
NO
176
COUPON EXPIRES

FIGURE 14

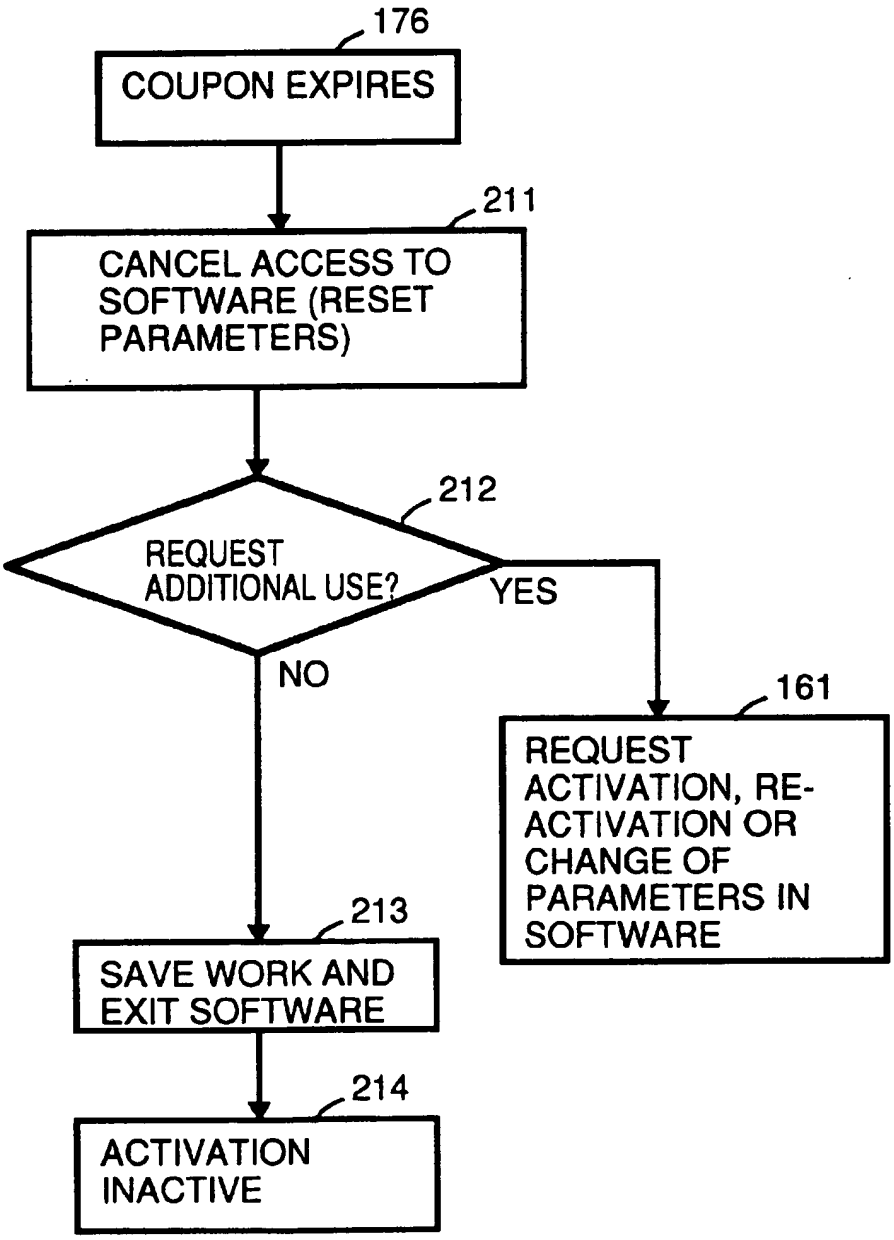

176
COUPON EXPIRES
211
CANCEL ACCESS TO SOFTWARE (RESET PARAMETERS)
212
REQUEST ADDITIONAL USE?
YES
NO
161
REQUEST ACTIVATION, RE-ACTIVATION OR CHANGE OF PARAMETERS IN SOFTWARE
213
SAVE WORK AND EXIT SOFTWARE
214
ACTIVATION INACTIVE

FIGURE 15

REMOTE INSTALLATION OF SOFTWARE BY A MANAGEMENT INFORMATION SYSTEM INTO A REMOTE COMPUTER

BACKGROUND

The present invention concerns management of computing devices and pertains particularly to the installation of software on a remote computing device.

Management information systems (MIS) are used to manage computing devices by monitoring and checking inventory, taking the current status of machine configurations (such as current memory configurations, hard drive capacity, RAM, CPU and other upgrades) as well as to monitor usage patterns. An MIS console **11** for an MIS system generally utilizes a desktop management interface (DMI) located within each personal computer (PC) or server. DMI is a standard interface which handles communication between management applications and all the manageable elements on or attached to a personal computer or server. DMI provides the communication between any management application and the manageable elements on a system. Within the DMI, the standard way of describing elements is provided by the management information format (MIF). The MIF is a prescribed grammar and syntax to an element's manageable attributes. MIF files are standard at the group level and at the element level, so common aspects of many different elements can be provided by using standard MIF files and MIF groups. Standard MIF files and groups exist for many common elements.

Within the DMI, a service layer is a program, running on the local machine or personal computer, that collects information from elements, manages that information in the MIF database, and passes the information to management applications as requested. The service layer controls communication between itself and management applications by means of a management interface (MI) and between itself and manageable elements by means of a component interface (CI). For example, a service layer interface for Windows 3.1X operating system is available from Intel Corporation, having a business address of 2200 Mission College Boulevard, Santa Clara, Calif. 95050.

Within the DMI, management applications are remote or local programs used for changing, interrogating, controlling, tracking and listing the elements of a system. A management application can be a local diagnostics or installation program, or a remote agent which redirects information from the DMI service layer over a network.

Manageable elements are hardware, software or peripherals that occupy or are attached to a personal computer or network server. For instance manageable elements include hard disks, word processors, CD-ROMs, printers, motherboards, operating systems, spreadsheets, graphics cards, sound cards, or modems. Each element provides information to the MIF database by means of an MIF file which contains the pertinent management information for that element. The information in the MIF file is compiled into the MIF database when the element is installed.

MIS managers can query individual machines to access DMIs and MIF databases on individual machines in order to obtain current information stored therein. Based on this information, MIS managers can schedule upgrades for outdated hardware and software configurations.

While an MIS has been used effectively to manage PCs and servers interconnected by a local area network (LAN), there has been no similar effective strategy to manage devices which are not connected to a LAN. For example, there is no effective strategy to manage portable computers which are seldom or never permanently connected to a LAN. Yet with the proliferation of the usage of portable computers, it is desirable to include such devices within an MIS.

SUMMARY OF THE INVENTION

In accordance with the preferred embodiment of the present invention, remote installation of an update of software is forwarded by a management information system into a remote computer. When a primary communication path between the management information system and the remote computer is unavailable and an alternate communication path is available, a connection is established between the management information system and the remote computer using the alternate communication path. The management information system requests from the remote computer the current version information about the software within the remote computer. When the management information system determines the current version of the software within the remote computer needs to be updated, the management information system determines whether the alternate communication path is adequate for downloading the update of the software. When the management information system determines that the alternate communication path is adequate for downloading the update of the software, the update of the software is downloaded from the management information system to the remote computer.

In the preferred embodiment, when the management information system determines that the alternate communication path is not adequate for downloading the update of the software, the download of the update of the software from the management information system to the remote computer is queued for performance when the computer is connected to the primary communication path or another alternative path.

For example, the primary communication path is through a local area network or alternately through a public telephone system or some other connection media, e.g., the internet. Likewise, for example, the alternate communication path is through a two-way paging network system, through a public telephone system, or through some other connection media.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows the connected relationship between a management information system (MIS) console and a portable computer in accordance with a preferred embodiment of the present invention.

FIG. **2** shows a management information system (MIS) which includes portable computers in accordance with a preferred embodiment of the present invention.

FIG. **3** shows an enhanced desktop management interface (DMI) which has been modified for use in a portable computer in accordance with a preferred embodiment of the present invention.

FIG. **4** shows an enhanced desktop management interface (DMI) which has been modified for use in a portable computer in accordance with an alternate preferred embodiment of the present invention.

FIG. **5** shows a table within an MIS console which is used to select a communication path to managed elements in accordance with a preferred embodiment of the present invention.

FIG. **6** shows a table within a personal computer which is used to select a communication path to an MIS console in accordance with a preferred embodiment of the present invention.

FIGS. 7A and 7B are a flowchart which illustrates remote code update of a remote computer in accordance with a preferred embodiment of the present invention.

FIGS. 8A and 8B are a flowchart which illustrates the establishment of communication with a remote computer in accordance with a preferred embodiment of the present invention.

FIGS. 9A and 9B are a flowchart which illustrates actions taken when a remote computer is reported lost or stolen in accordance with a preferred embodiment of the present invention.

FIG. 10 is a flowchart which illustrates the use of remote management to initialize software metering in accordance with a preferred embodiment of the present invention.

FIG. 11 is a flowchart which illustrates one type of software metering (periodic time style) initialized using remote management in accordance with a preferred embodiment of the present invention.

FIG. 12 is a flowchart which illustrates another type of software metering (counter style) initialized using remote management in accordance with a preferred embodiment of the present invention.

FIG. 13 is a flowchart which illustrates another type of software metering (timed use style) initialized using remote management in accordance with a preferred embodiment of the present invention.

FIG. 14 is a flowchart which illustrates another type of software metering (interactive style) initialized using remote management in accordance with a preferred embodiment of the present invention.

FIG. 15 is a flowchart which illustrates control of software metering when a coupon is expired in accordance with a preferred embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 shows a management information system (MIS) console 11 connected through an interconnection 10 to a portable computer 16. The dotted line between interconnection 10 and portable computer 16 indicates that portable computer 16 may be only intermittently available through interconnection 10. Interconnection 10 may include any combination of communications technology. For example, interconnection may include one, or a combination of, a local area network, a wide area network, the internet, the world wide web, a public telephone system, a private telephone system, a modem, a paging network system, radio frequency transmission, a cellular phone system, and so on.

For example, FIG. 2 shows a management information systems (MIS) which includes MIS console 11 and a network access 12. Network access 12 is, for example, a network server, a network provider, an internet access or a world wide web access. Network access 12 is connected through a local area network (LAN) 22 to a portable computer 20 and a computer 21. Based on the discretion of the user of portable computer 20, portable computer 20 may or may not be permanently attached to LAN 22. In addition to portable computer 20 and computer 21, other personal computers, servers and portable computers may be connected to LAN 22.

Network access 12 is connected to a telephone system 14 through a modem 13. A portable computer 17 and a portable computer 18 utilize telephone system 14 to connect network access 12 and, when connected are available to MIS console 11.

A paging network server 15 is connected to telephone system 14. Through two-way paging network server 15, MIS console 11 is able to contact portable computer 16. Use of two-way paging network server 15 has the advantage of making portable computer 16 always available to MIS console 11.

A portable computer 19 is not connected to any network and is thus currently unavailable to MIS console 11. Once portable computer 19 is connected to LAN network 22, telephone system 14 and/or paging network server 15, MIS console 11 will be able to access portable computer 19.

Enhanced Desktop Management Interface

FIG. 3 illustrates an enhanced desktop management interface (DMI) located within each of personal computers 16 through 20. Within the DMI, a service layer 30 collects information from elements, manages that information in an MIF database 33, and passes the information to management applications as requested.

Service layer 30 controls communication between itself and management applications by means of a management interface (MI) 31 and between itself and manageable elements by means of a component interface (CI) 32.

Management applications are remote or local programs for changing, interrogating, controlling, tracking and listing the elements of a system. A management application can be a local diagnostics or installation program, or a remote agent which redirects information from DMI service layer 30 over a network. For example, in FIG. 3, management interface 31 is shown to interface to a support management application 34, a DMI application 35, a LAN management application 36 and a setup program 37. Management interface 31 also interfaces to other management applications, as represented by an other management application 38.

Manageable elements are hardware, software or peripherals that occupy or are attached to a portable computer. For example, in FIG. 3, component interface 32 is shown interfacing with a word processor 41, a hard disk 42 and a CD-ROM 43. Component interface 32 also interfaces to other manageable elements, as represented by other element 45.

Within management information format (MIF) database 33 there is stored MIF files for the manageable elements and the management applications.

A communication management application 39, connected to management interface 31, and a communication device 44, connected to component interface 32 are added to the DMI in order to provide an alternate communication path for MIS. Communication device 44 is, for example, a two-way pager. Alternately, communication device 44 is a modem, a radio frequency transceiver or some other connection to a communication medium which allows the personal computer to establish contact with an MIS console 11. In some embodiments of the present invention, communication can be one-way, for example, implemented using a one-way pager. Communication through communication device 44 is controlled by communication management application 39.

The existence of an alternate communication path for MIS increases flexibility for an MIS manager. In effect, the existence of the alternate communication path for MIS allows an MIS manager to manage portable computers and other devices which are not directly connected to a LAN. The alternate communication path can be used, for example, by the MIS manager to direct a managed portable computer of the need to connect to a primary communication path in order to receive an update.

FIG. 4 illustrates an alternate structure for the enhanced desktop management interface (DMI) shown in FIG. 3. In FIG. 4, a communication management application **50** is connected to each of the management applications. For example, as shown in FIG. 4, communication management application **50** is connected to support management application **34**, DMI application **35**, LAN management application **36** and setup program **37**. Communication management application **50** also interfaces to other management applications, as represented by other management application **38**.

Communication management application **50** selects a communication path to MIS console **11**. For example, communication management application **50** uses an LAN interface **51**, a modem interface **52** or a two-way pager interface **53** to establish a communication path to MIS console **11**. Alternately, communication device **44** is a modem, or some other connection to a communication medium which allows the personal computer to establish contact with an MIS console **11**.

Within communication management application **50**, the available communication paths are priority ranked. For example, the communication management application **50** will communicate with MIS console **11** over LAN **22** when the personal computer is connected to LAN **22**. If communication management application **50** is not connected to an LAN with access to MIS console **11**, communication management application **50** will establish contact with MIS console **11** over public telephone system **14** using modem interface **52**. If communication management application **50** is not connected to an LAN with access to MIS console **11** and access over public telephone system **14** using modem interface **52** is not available, communication management application **50** will establish contact with MIS console **11** using two-way pager interface **53** to make contact with paging network server **15**.

Because the rate of data transfer between MIS console **11** and the personal computer depends upon which communication path is used, there are some transactions which are restricted, depending of the communication path. For example, when contact with MIS console **11** is established using two-way pager interface **53** to make contact through paging network server **15**, major downloads of software from MIS console **11** or major uploads of database data from MIS console **11** are not performed.

FIG. **5** shows a table **60** within MIS console **11** which lists various devices managed by MIS console **11**. For each managed device, in descending order of priority, the potential communication path types are listed. For each potential path type, an address, cost, throughput rate and expected response time are listed.

Specifically, as shown in FIG. **5**, a column 61 of table 60 lists the device by device number. In column 61, portable computer **16**, portable computer **17**, portable computer **18**, portable computer **19**, portable computer **20** and computer **21** are listed.

In a column 62 of table 60, various communication path types are listed for each listed device. For the example shown in FIG. **5**, each listed device can be contacted using one or more of the following path types: LAN, MODEM, PAGER. The path type of LAN indicates that MIS console **11** can contact the device through LAN **22** or some other LAN. The path type of MODEM indicates that MIS console **11** can contact the device through telephone system **14**. The path type of PAGER indicates that MIS console **11** can contact the device through paging network server **15**.

In a column 63 of table 60, the address of the device over the communication path is listed. This address is specific to the particular device.

In a column 64 of table 60, a cost code is listed which indicates for the listed device a cost for the connection. This cost code is used by MIS console **11** to determine whether, for a particular management operation, it is desirable to utilize this communication path to perform the management operation, depending upon urgency. In FIG. **5**, there are three cost codes listed. For example, the TIME DEP cost code indicates that the particular cost is time dependent. In this case, there is a very low cost for performing a management operation in off hours, and a higher cost for performing the management operation in normal working hours. Thus, unless it is important to perform a particular management operation immediately, MIS console **11** can elect to perform the management operation at a later time. The MODEM cost code is the cost code assigned when telephone system **14** is used to complete the call. Generally, the MODEM cost is higher than the TIME DEP cost. The PAGER cost code is the cost code assigned when paging network server **15** is used to complete the call. Generally, the PAGER cost is higher than the MODEM cost. Other alternative communication methods also can be ranked accordingly.

In a column 65 of table 60, a throughput rate is listed which indicates for the listed device a throughput rate for the particular communication path. For communication paths though telephone system **14**, the throughput rate can vary based on the throughput rate of a modem installed within the device.

In a column 66 of table 60, a response time is listed which indicates an expected response time for the device to respond when contacted through the particular communication path. Alternately, the listed response time indicates a maximum allowed response time for the device to respond when contacted through the particular communication path. When the device fails to respond, this indicates to MIS console **11** that the device is not available to be contacted through the chosen communication path. MIS console **11** can then choose to attempt connection through an alternate communication path for the device, try the same communication path at another time, or abandon the attempt to contact the device.

Likewise, the individual personal computers can similarly prioritize data communication paths for contacting MIS console **11** to initiate or respond to significant MIS events such as hardware failure warnings. Upon detection of a significant event, communications management application **50** can inform the user of the personal computer of the event or can automatically contact MIS console **11** through an available communication data path.

For example, FIG. **6** shows a table 70 within personal computer **18** which lists in descending order of priority, the potential communication path types to MIS console **11**. For each potential path type, an address, cost, throughput rate and expected response time are listed.

Specifically, as shown in FIG. **6**, in a column 72 of table 70, various communication path types are listed. For the example shown in FIG. **6**, personal computer **18** can contact MIS console 11 through the following path types: LAN, MODEM, PAGER.

In a column 73 of table 70, the address of MIS console **11** over the communication path is listed. This address is specific to MIS console **11**.

In a column 74 of table 70, a cost code is listed which indicates for the listed device a cost for the connection. This

cost code is used by portable computer to determine whether, for a particular management operation, it is desirable to utilize this communication path to perform the management operation, depending upon urgency. In FIG. 6, there are three cost codes listed, as further discussed above.

In a column 75 of table 70, a throughput rate is listed which indicates for the listed device a throughput rate for the particular communication path. For communication paths though telephone system 14, the throughput rate can vary based on the throughput rate of a modem installed within the device.

In a column 76 of table 70, a response time is listed which indicates an expected response time for the device to respond when contacted through the particular communication path. Alternately, the listed response time indicates a maximum allowed response time for the device to respond when contacted through the particular communication path. When the device fails to respond, this indicates to portable computer 18 that MIS console 11 is not available to be contacted through the chosen communication path. Portable computer 18 can then choose to attempt connection to MIS console 11 through an alternate communication path, try the same communication path at another time, or abandon the attempt to contact MIS console 11.

Using alternate communication paths to manage portable computers and other devices offer some significant advantages. For example, code updates can be down loaded to devices even when they are not physically attached to any network. Additionally, management access to a computing device is a significant security feature. For example, sensitive data can be erased from a lost or stolen computing device. Similarly, a lost or stolen computing device can be instructed not to boot up. Also, the alternate communication path could be used as an alarm to indicate to a manager that the computing device has been disconnected from a network without authorization.

Also, the alternate communication path can be utilized as a low speed network for data communication, even allowing e-mail or low-throughput connection to the internet.

Remote Code Update/Installation

FIGS. 7A and 7B are a flowchart which illustrates remote code update of a remote computer in accordance with a preferred embodiment of the present invention. The same process may be used for installation of software on a remote computer. A left half 79 of the flow chart shows the activity of an MIS system. A right half 80 of the flow chart shows the activity of a remote computer.

In a step 81, the MIS system initiates a request to verify software versions on the remote computer. In a step 82, the MIS system establishes communication with the remote computer. In a step 83, the remote computer receives a valid message request. In a step 84, an internal check of the remote computer is made to determine whether the host processor within the remote computer is powered up. In a step 85, if the remote computer is asleep, it is awakened. In a step 86, the remote computer acknowledges to the MIS system that the remote computer is connected and ready for further transmissions.

In a step 87, the MIS system determines whether there is a valid connection. If there is not a valid connection, in a step 88, the MIS system queues the request for a later attempt. If in step 87, the MIS system determines there is a valid connection, in a step 89, the MIS system requests from the remote computer the version number of the image of the software on the hard drive of the remote computer. In a step 90, the remote computer processes the requests and forwards the requested data to the MIS system. In a step 91 the data is received from the remote computer by the MIS system.

In a step 92, the MIS system determines whether the image needs upgrading. If upgrading is needed, in a step 93, the MIS system determines whether the connection is appropriate for the action. If the connection is appropriate for the action, in a step 94, the MIS system begins to down load the new files to the remote computer. In a step 99 the remote computer downloads the files into separate area checks. In a step 100, the remote computer displays a note to the user about the results of the action. In a step 101, the remote computer acknowledges the operation to the MIS system. In a step 102, the remote computer requests the user to close files and reboot the remote computer.

If in step 93, the MIS system determines that the connection is not appropriate for the action, in a step 95, the MIS system leaves a note to request appropriate correction of user through. an appropriate connection. In a step 96, the MIS system queues a remote request for the next appropriate connection.

When the MIS system is ready to close the connection, in a step 97, the MIS system updates its data base information on the customer. In a step 98, the MIS system removes the communication link with the remote computer. In a step 103, the remote computer removes its communications link with the MIS system.

Establishing Connection with a Remote Computer

FIGS. 8A and 8B are a flowchart which illustrate the establishment of communication with a remote computer in accordance with a preferred embodiment of the present invention. A left half 110 of the flow chart shows the activity of an MIS system. A right half 111 of the flow chart shows the activity of a remote computer.

In a step 112, the MIS system searches the LAN to which the MIS system is connected in order to determine whether the target remote computer is connected to that LAN. In a step 113, the MIS system determines whether the remote computer is connected to the LAN. If the remote computer is connected to the LAN, in a step 114, the MIS system establishes connection with the remote computer. In a step 115, the remote computer acknowledges the connection. In a step 116, the connection is completed.

If in step 113, the MIS system determines the remote computer is not connected to the LAN, in a step 117, the MIS system initiates an alternative path to the remote computer which requests the status of the remote computer and the various connection paths or types which could be used by the MIS system to connect to the remote computer. For example, the alternative path could be a page or a telephone connection. In a step 118, the remote computer responds with its current connection capabilities.

In a step 119 the MIS system selects the appropriate type connection for the particular activity required. In a step 120, the MIS system determines whether the appropriate connection type is an LAN. If so, in a step 121, the MIS system establishes a link profile. The link profile indicates, for example, whether the link is fast, has a small delay, or is inexpensive. In a step 122, the remote computer establishes an LAN connection with the MIS system. In a step 123, the MIS system determines whether the appropriate connection type is a telephone system. If so, in a step 124, the MIS system establishes a link profile. The link profile indicates, for example, whether the link is fast, has a small delay, or is inexpensive. In a step 125, the remote computer establishes a telephone connection with the MIS system.

In a step **126**, the MIS system determines whether the appropriate connection type is a pager. If so, in a step **127**, the MIS system establishes a link profile. The link profile indicates, for example, whether the link is fast, has a small delay, or is inexpensive. In a step **128**, the remote computer determines whether reception is acceptable to establish the link. If reception is unacceptable, in a step **130**, the remote computer denies the request for a link.

When a link has been established in step **122**, **125** or **128**, in a step **132**, the remote computer verifies to the MIS system that the connection is established. In a step **133**, the connection is complete. As will be understood by persons of ordinary skill in the art, in addition to the connection types shown, other connection technologies may also be used to establish a link.

When there is no appropriate type of link available for connection, in a step **129**, the MIS system queues requests for the remote computer for a later attempt. In a step **131**, the MIS system notes that the connection is incomplete.

Computer Security

FIGS. **9A** and **9B** are a flowchart which illustrate actions taken when a remote computer is reported lost or stolen in accordance with a preferred embodiment of the present invention. A left half **140** of the flow chart shows the activity of an MIS system. A right half **141** of the flow chart shows the activity of a remote computer.

In a step **142**, the computer is reported stolen or lost by the owner. In a step **143**, a verification of the report is made to determine the legitimacy of the report. Steps **142** and **143** is performed, for example, by an operator or manager of the MIS system. In a step **144**, the MIS system establishes communication with the remote (stolen or lost) computer. In a step **145**, the remote computer receives a valid message request. In a step **146**, an internal check of the remote computer is made to determine whether the host processor within the remote computer is powered up? In a step **147**, if the remote computer is asleep, it is awakened. In a step **148**, the remote computer acknowledges to the MIS system that the remote computer is connected and ready for further transmissions.

In a step **149**, the MIS system determines whether there is a valid connection. If there is not a valid connection, in a step **150**, the MIS system queues the request for a later attempt. If in step **149**, the MIS system determines there is a valid connection, in a step **151**, the MIS system initiates a command to lock the computer and/or to encrypt or erase the data of the user. In a step **152** the remote computer responds by locking the computer and/or encrypting or erasing the data of the user. In a step **153**, the remote computer acknowledges the command has been completed. In a step **154**, the MIS system removes the connection. In a step **155**, the remote computer disconnects the link.

Software Metering

FIGS. **10** through **15** illustrate flow for various types of software metering which can be done using remote management. Software metering provides for limits, of one sort or another, on the use of software. As illustrated below, various types of metering can be used. The flexibility allows for various applications such as, for example, timed examinations, games with various levels and trial periods.

FIG. **10** is a flowchart which illustrates the use of remote management to initialize software metering in accordance with a preferred embodiment of the present invention. In a step **161**, a request activation, re-activation or change of parameters in software is made. This can be initiated by a user using the computer system. Alternatively, as illustrated by a step **164**, a remote management entity (e.g., a manager of information systems or an instructor) can initiate a new parameter session. The remote management entity initiates change, for example, when it is necessary to update computer systems where a site license has superseded terms for individual licenses.

Once a the request for activation, re-activation or change of parameters in software is made, in a step **162**, the terms of the activation, re-activation or change of parameters is agreed upon by the computer system and the remote management. The terms include, for example, method of payment and the style of software metering which is to be used. Once an agreement on terms is reached, in a step **163**, a user of the computer system is notified, for example, by an activation message or an activation page being sent to the computing system and appropriately displayed or made available to the user. In a step **163** a coupon is activated in accordance with the terms agreed upon in step **162**.

Various styles of software metering may be used, as illustrated by the flowcharts in FIGS. **10**, **11**, **12** and **13**. FIG. **11** is a flowchart which illustrates a style of software metering which allows usage of software for a set period of time. This style of software metering is useful, for example, to allow a user a trial period for software. In this style of software metering, after the coupon is activated in step **165**, in a step **171**, an end date is set after which a user will not be allowed to access the software. In a step **172**, when a user attempts to access the software, in a step **173**, a check is made to see whether the current date is the end date or after the end date. If the current date is not the end date or after the end date, access to the software is allowed to continue. After a set time of delay, as illustrated in a step **175**, the current date is checked again in step **173**, to check whether the current date is the end date or after the end date.

When in step **173**, it is determined that the current date is the end date or after the end date, in a step **174**, it is recognized that the end date has arrived. In a step **176**, the coupon allowing usage of the software is expired. This is handled as is illustrated by FIG. **15** below.

FIG. **12** is a flowchart which illustrates a style of software metering which allows usage of software for a set number of times. This style of software metering is useful, for example, to allow a user to test software on a trial basis. In this style of software metering, after the coupon is activated in step **165**, in a step **181**, a counter parameter is set. After the count has been met, a user will not be allowed to access the software. In a step **182**, when a user uses the computer, if it is detected in a step **183**, that the user has accessed the software, the counter is decremented (or incremented depending upon the implementation). In a step **185**, a check is made to see whether the counter parameter has reached the final count (in the example shown in FIG. **12**, the final count is 0). If the counter parameter has not reached the final count, the user is allowed to utilized the software. In a step **187**, user activity is monitored to determine when the software has been exited. The software is exited, the flow returns to step **182**.

If in step **185**, the check indicates the counter parameter has reached the final count, in a step **176**, the coupon allowing usage of the software is expired. This is handled as is illustrated by FIG. **15** below.

FIG. **13** is a flowchart which illustrates a style of software metering which allows usage of software for a set duration

of time. This style of software metering is useful, for example, for examinations, trials of software or for game applications. In this style of software metering, after the coupon is activated in step **165**, in a step **191**, a duration parameter is set which indicates the total access time allowed without further accesses to a host management system. The current accumulated time is also initialized to zero. After the duration of time has been met, a user will not be allowed to access the software. In a step **192**, when a user uses the computer, if it is detected in a step **193**, that the user has accessed the software, a time stamp is used to record the starting (current) time use of the software begins.

In a step **195**, when a user uses the computer, if it is detected in a step **196**, that the user has exited the software, in a step **197**, the accumulated time is increased by the amount of time the software was used. As illustrated by FIG. **13**, the amount the accumulated time is increased is the different between the current time and the begin time stored in step **194**. After calculation of accumulated time in step **197**, step **192** is repeated.

If in step **196**, it is determined that the user has not exited the software, in a step **198** a check is made as to whether the duration of time has expired (i.e., accumulated time+(current time—begin time)>=Total access time (D)). If the duration time has not expired, in a step **199**, a delay is taken, depending on required accuracy, before checking for user activity in a step **195**.

If, in step **198**, the check indicates the duration of time has expired, in a step **176**, the coupon allowing usage of the software is expired. This is handled as is illustrated by FIG. **15** below.

FIG. **14** is a flowchart which illustrates a style of software metering which allows usage of software for a interaction level. This style of software metering is useful, for example, for game applications. In this style of software metering, after the coupon is activated in step **165**, in a step **201**, certain access levels are authorized. These indicate which access levels may be utilized by a user. In a step **202**, when a user uses the computer, if it is detected in a step **203** that the user has not completed an access level, the user is allowed continued access to the software.

If it is detected in a step **203** that the user has completed an access level, in a step **204**, a check is made to determine whether the user is authorized to use the next level. If it is determined in step **204** that the user is authorized to use the next level, the user is allowed continued access to the software and returns to step **202**.

If it is determined in step **204** that the user is not authorized to use the next level, in step **176**, the coupon allowing usage of the software is expired. This is handled as is illustrated by FIG. **15** below.

FIG. **15** is a flowchart which illustrates what happens when in step **176**, the coupon allowing usage of the software is expired. In a step **211**, access to the software is canceled and the parameters are reset. In a step **212**, the user is queried as to whether additional use is requested. If in step **212**, the user indicates no additional use is desired, in a step **213**, the work is saved for later access by the user or by the remote management system and the software is exited. In a step **214**, the activation status is placed as inactive, which will prevent further access to the software.

If in step **212**, the user indicates additional use is desired, control is returned to step **161**, shown in FIG. **10**. In step **161**, a request activation, re-activation or change of parameters in software is made.

The foregoing discussion discloses and describes merely exemplary methods and embodiments of the present invention. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

We claim:

1. A method for remote installation of an update of software by a management information system into a remote computer comprising the following steps:

(a) when a primary communication path between the management information system and the remote computer is unavailable and an alternate communication path is available, establishing connection between the management information system and the remote computer using the alternate communication path;

(b) requesting, by the management information system from the remote computer information, current version information of the software within the remote computer; and,

(c) when the management information system determines a current version of the software within the remote computer needs to be updated, performing the following substeps:

(c.1) determining by the management information system whether the alternate communication path is adequate for downloading the update of the software,

(c.2) when the management information system determines in step (c.1) that the alternate communication path is adequate for downloading the update of the software, downloading the update of the software from the management information system to the remote computer,

(c.3) when the management information system determines in step (c.1) that the alternate communication path is not adequate for downloading the update of the software, refraining from downloading the update of the software from the management information system to the remote computer, and

(c.4) when the management information system determines in step (c.1) that the alternate communication path is not adequate for downloading the update of the software, queuing the download of the update of the software from the management information system to the remote computer for performance when the computer is connected to the primary communication path.

2. A method as in claim **1** wherein in step (a) the primary communication path is through a local area network.

3. A method as in claim **1** wherein in step (a) the alternate communication path is through a two-way paging network system.

4. A method as in claim **1** wherein in step (a) the alternate communication path is through a public telephone system.

5. A method as in claim **1** wherein in step (a) the primary communication path is through a public telephone system.

6. A method as in claim **1** wherein in step (a) the primary communication path is through an internet system.

7. A method for remote installation of an update of software by a management information system into a remote computer comprising the following steps:

(a) establishing a connection link between the management information system and the remote computer;

(b) obtaining, by the management information system from the remote computer, current version information about the software;

(c) when the management information system determines the software on the remote computer requires updating, performing the following substeps:

(c.1) determining by the management information system whether the connection link is adequate for downloading an update of the software,

(c.2) when the management information system determines in step (c.1) that the connection link is adequate for downloading the update of the software, downloading the update of the software from the management information system to the remote computer,

(c.3) when the management information system determines in step (c.1) that the connection link is not adequate for downloading the update of the software, refraining from downloading the update of the software from the management information system to the remote computer, and

(c.4) when the management information system determines in step (c.1) that the connection link is not adequate for downloading the update of the software, queuing the download of the update of the software from the management information system to the remote computer for performance when the computer is connected to a second connection link which is adequate for downloading the update of the software.

8. A method as in claim **7** wherein in substep (c.4) the second connection link is through a local area network.

9. A method as in claim **7** wherein in substep (c.4) the second connection link is through a public telephone system.

10. A method as in claim **7** wherein in step (a) the connection link is through a two-way paging network system.

11. A method as in claim **7** wherein in step (a) the connection link is through a public telephone system.

12. A method as in claim **7** wherein in step (a) the connection link is through an internet system.

13. A method as in claim **7** wherein step (a) includes when the remote computer is not currently powered up, awakening the remote computer.

14. Storage media which stores a software application which performs a method for remote installation of an update of software by a management information system into a remote computer, the method comprising the following steps:

(a) establishing a connection link between the management information system and the remote computer;

(b) obtaining, by the management information system from the remote computer, current version information about the software;

(c) when the management information system determines the software on the remote computer requires updating, performing the following substeps:

(c.1) determining by the management information system whether the connection link is adequate for downloading an update of the software,

(c.2) when the management information system determines in step (c.1) that the connection link is adequate for downloading the update of the software, downloading the update of the software from the management information system to the remote computer,

(c.3) when the management information system determines in step (c.1) that the connection link is not adequate for downloading the update of the software, refraining from downloading the update of the software from the management information system to the remote computer, and

(c.4) when the management information system determines in step (c.1) that the connection link is not adequate for downloading the update of the software, queuing the download of the update of the software from the management information system to the remote computer for performance when the computer is connected to a second connection link which is adequate for downloading the update of the software.

15. Storage media as in claim **14** wherein in substep (c.4) the second connection link is through a local area network.

16. Storage media as in claim **14** wherein in substep (c.4) the second connection link is through a public telephone system.

17. Storage media as in claim **14** wherein in substep (c.4) the second connection link is through an internet system.

18. Storage media as in claim **14** wherein in step (a) the connection link is through a two-way paging network system.

19. Storage media as in claim **14** wherein in step (a) the connection link is through a public telephone system.

20. Storage media as in claim **14** wherein step (a) includes when the remote computer is not currently powered up, awakening the remote computer.

21. Storage media as in claim **14** wherein step (a) includes when the remote computer is not currently powered up, awakening the remote computer.

22. A method for remote installation of software by a management information system into a remote computer comprising the following steps:

(a) when a primary communication path between the management information system and the remote computer is unavailable and an alternate communication path is available, establishing connection between the management information system and the remote computer using the alternate communication path;

(b) when the management information system determines to download software, performing the following substeps:

(b.1) determining by the management information system whether the alternate communication path is adequate for downloading the software,

(b.2) when the management information system determines in step (c.1) that the alternate communication path is adequate for downloading the software, downloading the software from the management information system to the remote computer,

(b.3) when the management information system determines in step (b.1) that the alternate communication path is not adequate for downloading the software, refraining from downloading the software from the management information system to the remote computer, and

(b.4) when the management information system determines in step (b.1) that the alternate communication path is not adequate for downloading the software, queuing the download of the software from the management information system to the remote computer for performance when the computer is connected to the primary communication path.

* * * * *

US005860012A

United States Patent [19]

Luu

[11] Patent Number: 5,860,012

[45] Date of Patent: Jan. 12, 1999

[54] **INSTALLATION OF APPLICATION SOFTWARE THROUGH A NETWORK FROM A SOURCE COMPUTER SYSTEM ON TO A TARGET COMPUTER SYSTEM**

[75] Inventor: **Linda Luu**, Beaverton, Oreg.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **859,277**

[22] Filed: **May 19, 1997**

Related U.S. Application Data

[63] Continuation of Ser. No. 591,222, Jan. 18, 1996, abandoned, which is a continuation of Ser. No. 130,097, Sep. 30, 1993, abandoned.

[51] **Int. Cl.**[6] **G06F 11/30**

[52] **U.S. Cl.** **395/712**; 395/701; 395/200.5

[58] **Field of Search** 395/701, 712, 395/200.5

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,021,949	6/1991	Morten et al.	395/200
5,086,502	2/1992	Malcolm	395/575
5,257,378	10/1993	Sideserf et al.	395/700
5,265,239	11/1993	Ardolino	395/500
5,361,358	11/1994	Cox et al.	395/700
5,386,564	1/1995	Shearer et al.	395/650
5,388,211	2/1995	Hornbuckle	395/200

OTHER PUBLICATIONS

Mamram, "Maintenance . . . Mainframes", May 1991, pp. 113–119.

T. Busse, "WinInstall installs Windows apps from central location", INFOWORLD, p. 54, May 24, 1993.

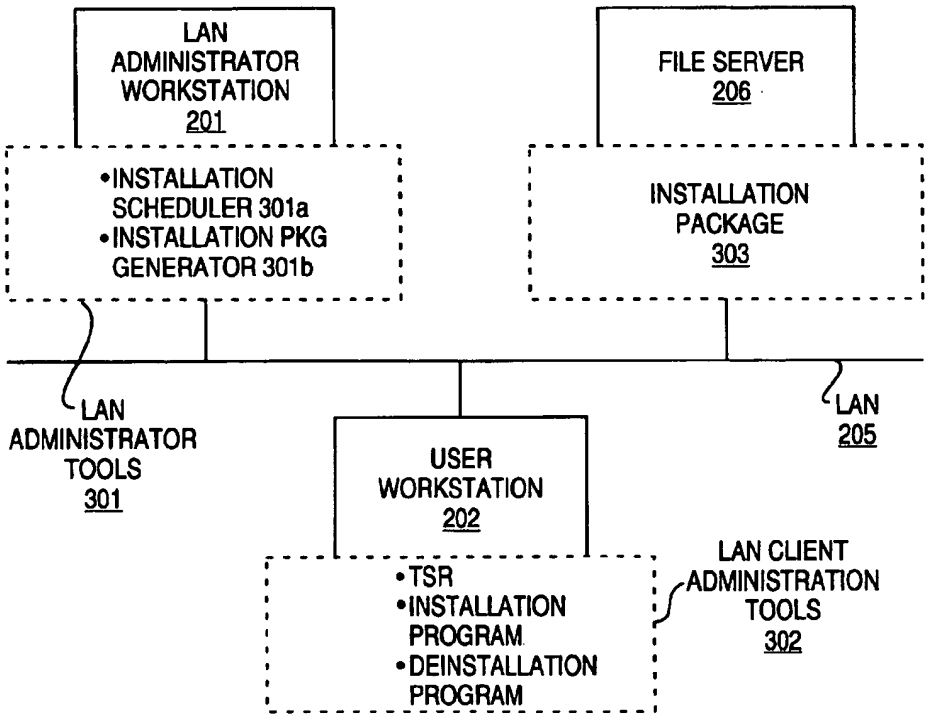
"WinInstall Sends Applications As E-Mail Attachments", PC MAGAZINE, p. 66, Jul. 1993.

Primary Examiner—Larry D. Donaghue
Assistant Examiner—John Follansbee
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman LLP

[57] **ABSTRACT**

A technique for the remote installation of application software from a source computer system to one or more target computer systems (workstation) coupled to a Local Area Network (LAN). The present invention allows a LAN Administrator to install application software on a user's workstation automatically at any time without user's intervention. The state of (i.e. a snapshot of) the LAN Administrator's system before and after the installation of the application software is captured and an installation package is built. Installation on the user workstations is then scheduled. For installation, the installation package is transmitted to the user workstation where an install program carries out commands in the installation package for installing the application software.

35 Claims, 6 Drawing Sheets



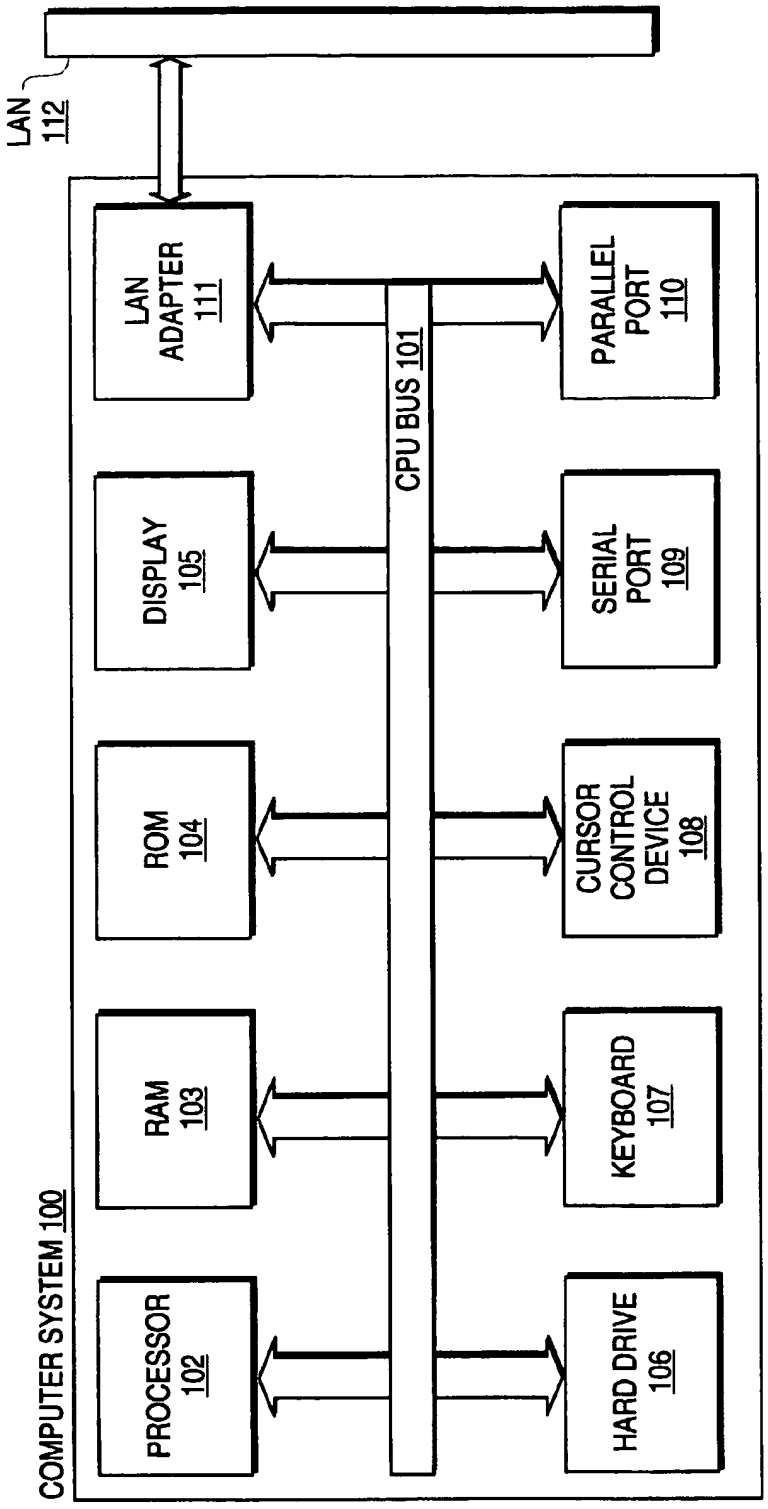

LAN
112
LAN ADAPTER 111
PARALLEL PORT 110
CPU BUS 101
DISPLAY 105
SERIAL PORT 109
ROM 104
CURSOR CONTROL DEVICE 108
RAM 103
KEYBOARD 107
PROCESSOR 102
HARD DRIVE 106
COMPUTER SYSTEM 100

FIG. 1

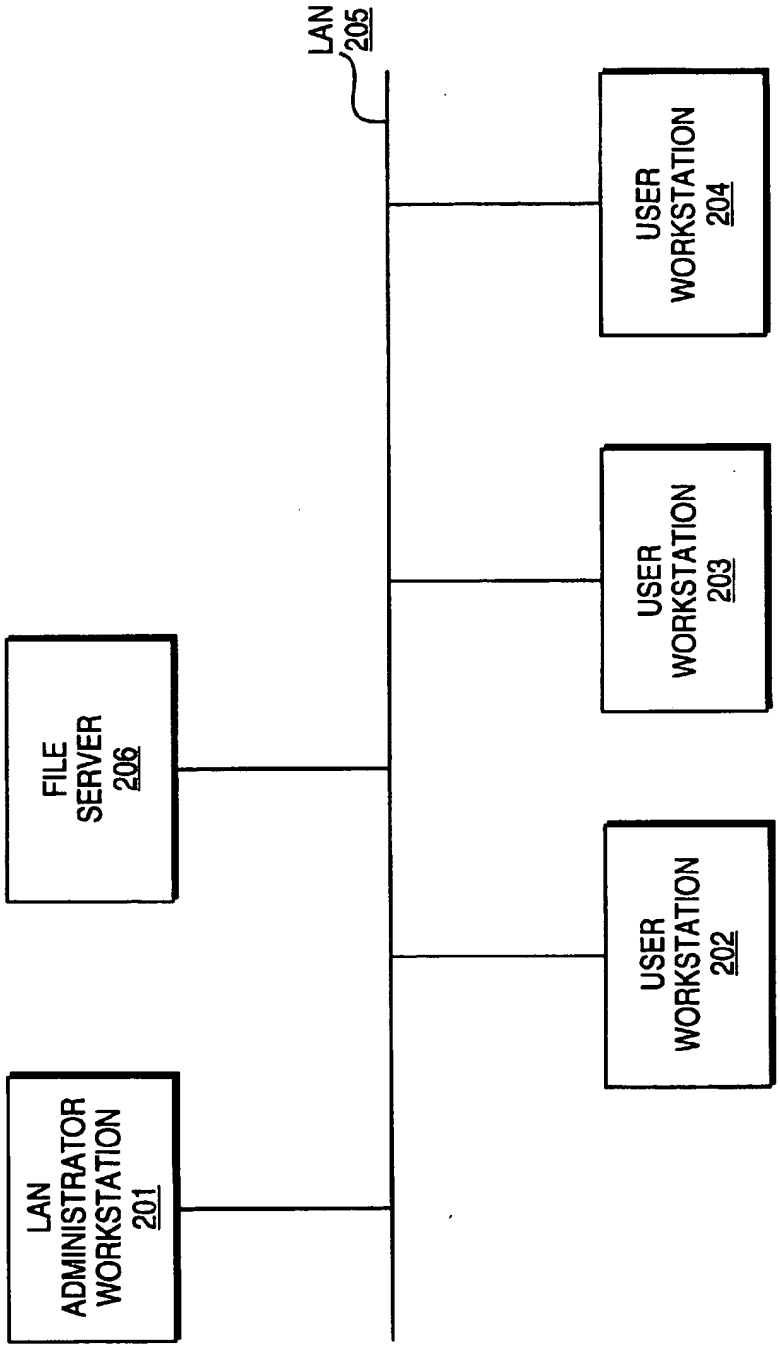

LAN
205
USER WORKSTATION
204
USER WORKSTATION
203
FILE SERVER
206
USER WORKSTATION
202
LAN ADMINISTRATOR WORKSTATION
201

FIG. 2

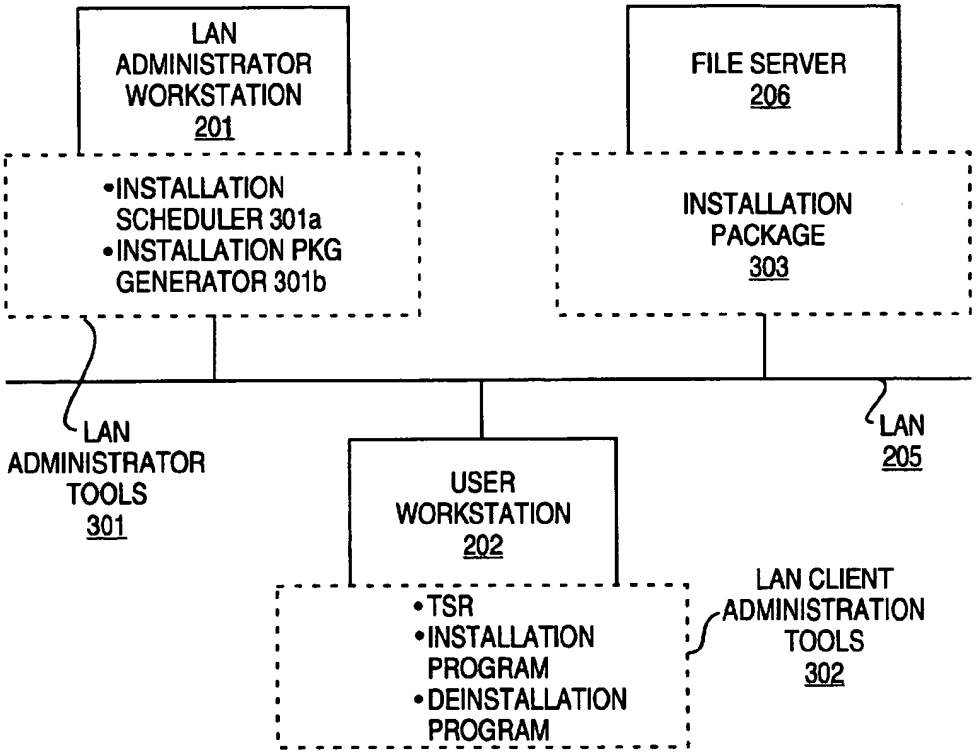

LAN
ADMINISTRATOR
WORKSTATION
201
•INSTALLATION
SCHEDULER 301a
•INSTALLATION PKG
GENERATOR 301b
FILE SERVER
206
INSTALLATION
PACKAGE
303
LAN
ADMINISTRATOR
TOOLS
301
LAN
205
USER
WORKSTATION
202
•TSR
•INSTALLATION
PROGRAM
•DEINSTALLATION
PROGRAM
LAN CLIENT
ADMINISTRATION
TOOLS
302

FIG. 3

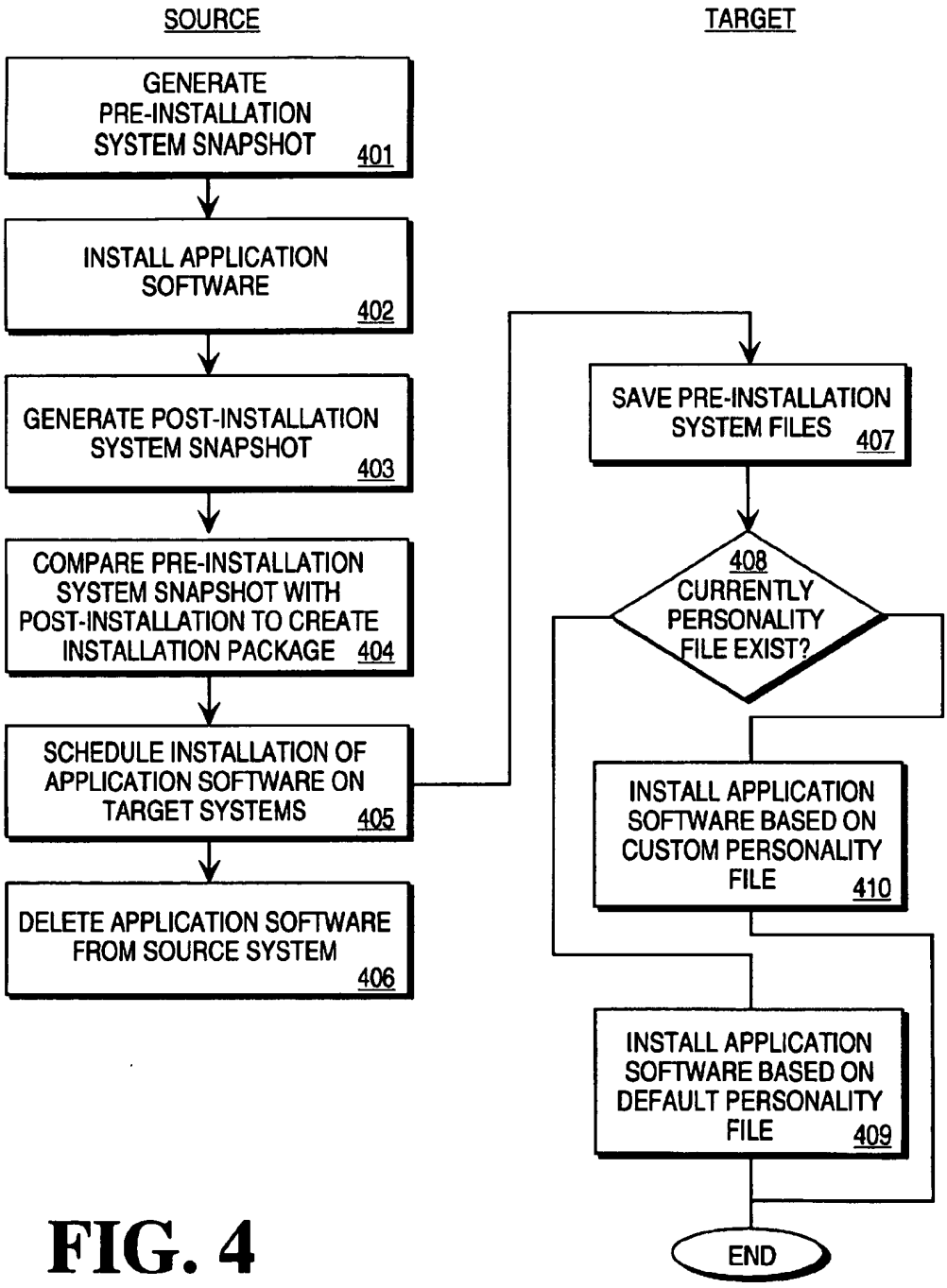

SOURCE
TARGET
GENERATE PRE-INSTALLATION SYSTEM SNAPSHOT 401
INSTALL APPLICATION SOFTWARE 402
GENERATE POST-INSTALLATION SYSTEM SNAPSHOT 403
COMPARE PRE-INSTALLATION SYSTEM SNAPSHOT WITH POST-INSTALLATION TO CREATE INSTALLATION PACKAGE 404
SCHEDULE INSTALLATION OF APPLICATION SOFTWARE ON TARGET SYSTEMS 405
DELETE APPLICATION SOFTWARE FROM SOURCE SYSTEM 406
SAVE PRE-INSTALLATION SYSTEM FILES 407
408 CURRENTLY PERSONALITY FILE EXIST?
INSTALL APPLICATION SOFTWARE BASED ON CUSTOM PERSONALITY FILE 410
INSTALL APPLICATION SOFTWARE BASED ON DEFAULT PERSONALITY FILE 409
END

FIG. 4

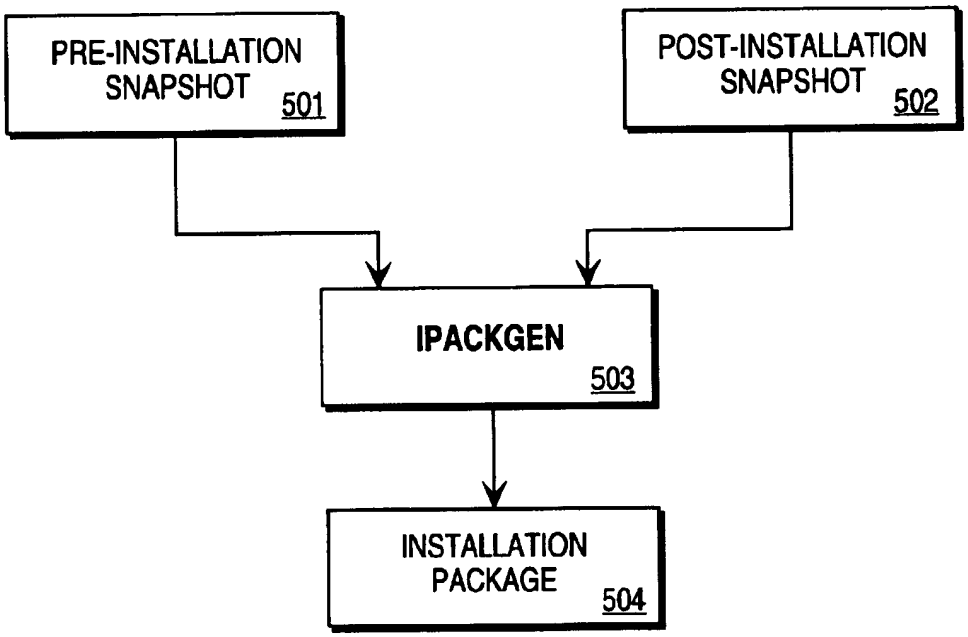

PRE-INSTALLATION SNAPSHOT 501
POST-INSTALLATION SNAPSHOT 502
IPACKGEN 503
INSTALLATION PACKAGE 504

FIG. 5

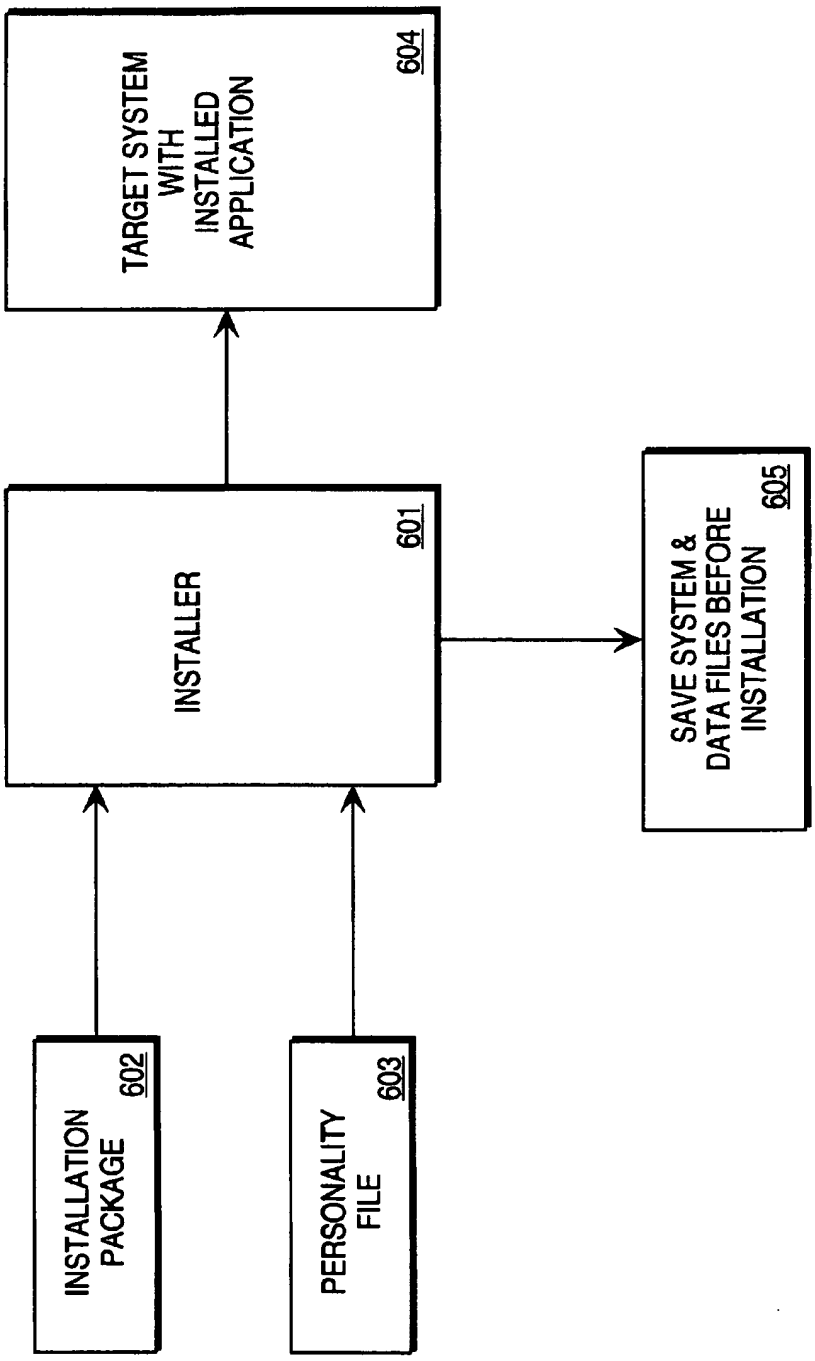

TARGET SYSTEM WITH INSTALLED APPLICATION
604
INSTALLER
601
SAVE SYSTEM & DATA FILES BEFORE INSTALLATION
605
INSTALLATION PACKAGE
602
PERSONALITY FILE
603

FIG. 6

INSTALLATION OF APPLICATION SOFTWARE THROUGH A NETWORK FROM A SOURCE COMPUTER SYSTEM ON TO A TARGET COMPUTER SYSTEM

This is a continuation of application Ser. No. 08/591,222, filed Jan. 18, 1996, now abandoned, which is a continuation of application Ser. No. 08/130,097, filed Sep. 30, 1993, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to the field of management of computer systems on a network, and specifically to installation of application software on computer systems through a network.

2. Prior Art

Local Area Networks (LANs) have been developed for interconnecting computer systems for communication amongst computer users, for the exchange of data and for the sharing of resources (e.g. printers, facsimile devices, modems and the like). To properly implement and manage a LAN, a LAN administrator is needed. A LAN Administrator's duties may include such functions as registering users to use the Local Area Network, maintaining the shared resources and monitoring the network load. LANs may also be used to manage the computer systems connected to the network.

With respect to management of the computer systems on the LAN, a desirable function for the LAN administrator is to remotely install application software on systems. Moreover, it would be desirable to perform such installations without the presence of the LAN Administrator. However, known techniques for remote installation of software require the writing of scripts outlining the installation procedures. The writing of scripts is a manual, error prone task which must be performed for each system onto which an application is to be installed.

An improved implementation for remote installation of applications by a LAN administrator is the netOctopus™ program, available from B&L Impuls Software GmbH. netOctopus operates in an Apple® Macintosh networked environment. netOctopus provides for the remote execution of installation scripts on systems in which the application software is to be installed. netOctopus utilizes the installer program that accompanies each Apple Macintosh system. However netOctopus merely allows the LAN Administrator to perform the installation across the network, as if they were sitting at the workstation themselves.

Consequently, it would be desirable to provide a means for automating script generation for remote installation of application software by a LAN Administrator. Further, it would be desirable to provide for such remote installation in a manner that does not require the presence of the LAN Administrator.

SUMMARY

A technique for the remote installation of application software from a source computer system to one or more target computer systems (workstations) coupled to a Local Area Network (LAN) is disclosed. The present invention allows a LAN Administrator to install application software on a user's workstation automatically at any time without user's intervention. The state of (i.e. a snapshot of) the LAN Administrator's system before and after the installation of the application software is captured and an installation package is built. Installation on the user workstations is then scheduled. At that time the installation package is transmitted to the user's workstation where a program carries out commands in the installation package for installing the application software.

The installation package consists of an IPACK format file and the files contained in an application software program. The IPACK format file contains sets of commands that are used to modify system files and perform other functions necessary to the installation of the application software. Further, a personality file may be defined which allows for custom tailoring of the installation on a user's workstation. Further, a UPACK format file provides instructions for deinstalling application software. Deinstallation of application software is necessary for removing unwanted or outdated applications from the user's workstation.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** is a block diagram of a computer system which may be utilized as a LAN administrator workstation or user workstation in the currently preferred embodiment of the present invention.

FIG. **2** is block diagram of a LAN with a LAN Administrator workstation and user workstations as may be implemented in the currently preferred embodiment of the present invention.

FIG. **3** is a block diagram illustrating the architecture of the main components of the LAN administrator workstation and a user workstation for implementation of the currently preferred embodiment of the present invention.

FIG. **4** is a flow chart illustrating the steps performed by the LAN Administrator workstation (source system) and the user workstation (target system) in performing remote application software installations in the currently preferred embodiment of the present invention.

FIG. **5** is a block diagram illustrating the inputs and outputs for generating an installation package on a LAN Administrator workstation as may be performed in the currently preferred embodiment of the present invention.

FIG. **6** is a block diagram showing the inputs and outputs for installing an application package on a user workstation as may be performed in the currently preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A technique for the remote installation of application software on user workstations connected to a Local Area Network (LAN) from a LAN Administrator workstation is described. In the following description, numerous specific details are set forth such as the network topology, in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without such specific details. In other instances, specific implementation details such as the steps for installation of any particular application software package have not been shown in detail in order not to unnecessarily obscure the present invention.

The currently preferred embodiment of the present invention is implemented for operation in networking environments utilizing the NetWare Network Operating System, available from the Novell Corporation of Provo, Utah. The respective workstations coupled to the network of the currently preferred embodiment would be executing the

MS/DOS and the Windows operating system environments, both available from Microsoft Corporation of Seattle, Washington. However, these implementations are not meant to be limiting as to the scope of the present invention. It would be apparent to one skilled in the art to practice the present invention in environments utilizing different network control operating systems and with workstations executing under non MS/DOS or non Windows operating systems.

Further, the following description uses the term "workstation" with respect to any computer system that a user or LAN Administrator may be using. The term "workstation" is not meant to indicate or denote any particular class of computer systems. Finally, the following description will make reference to certain data files and structures, e.g. a CONFIG. SYS file, that are well known to those familiar with the MS/DOS Operating System. Thus, further description of such data files and structures is not deemed necessary.

Overview of a Computer Systems in the Currently Preferred Embodiment

A computer system embodying a workstation of the currently preferred embodiment is described with reference to FIG. **1**. A computer system **100** as may be utilized by the currently preferred embodiment generally comprises a bus structure or other communication means **101** for communicating information between the various components of the computer system, a processor means **102** coupled with said bus **101** for processing information, a random access memory (RAM) or other storage device **103** (commonly referred to as a main memory) coupled with said bus **101** for storing information and instructions for said processor **102**, a read only memory (ROM) or other static storage device **104** coupled with said bus **101** for storing static information and instructions for said processor **102**, a display monitor **105** coupled with said bus **101** for displaying textual, graphical and image data generated by the computer system, a data storage device **106**, such as a magnetic disk and disk drive, coupled with said bus **101** for storing information and instructions, an alphanumeric input device **107** including alphanumeric and other keys coupled to said bus **101** for communicating information and command selections to said processor **102**, a cursor control device **108**, such as a mouse, trackball, cursor control keys, etc., coupled to said bus **101** for communicating information and command selections to said processor **102** and for controlling cursor movement. Additionally, the system will typically include, one or more ports for receiving input signal data. Such ports are illustrated here as serial port **109** and parallel port **110**.

The computer system **101** further includes a Local Area Network (LAN) adapter **111** for attaching to a LAN **112**. The LAN Adapter **111** is coupled to CPU bus **101** and is used to transmit/receive information to/from the LAN **112**.

It should further be noted that the processor **102** in the computer system would perform various of the processing functions, e.g. generation of the application installation package (LAN Administrator workstation), and actual installation of the application software (user workstation), which is described herein.

LAN Environment

FIG. **2** illustrates a local area networking environment of the currently preferred embodiment of the present invention. In the currently preferred embodiment, the LAN **205** is an Ethernet LAN. As mentioned above, the network control operating system is Novell NetWare. Referring back to FIG. **2**, a LAN Administrator workstation **201** is coupled to the LAN **205**. A plurality of user workstations **202–204** are also coupled to the LAN **205**. In operation, the LAN Administrator workstation **201** performs various functions in keeping the network and each of the individual user workstations operable. The present invention provides a means by which a LAN administrator can remotely schedule and cause the installation of application software on each of the user workstations **202–204**. Without such a facility, the LAN administrator would have to physically install the application software on each of the user workstations **202–204**. This is very inefficient since it may require "downtime" of the user workstation during work hours, thus causing a loss in productivity of the user. The alternative, having the application software installed during nonwork hours is also undesirable since it still requires the physical presence of the LAN administrator.

Further illustrated in FIG. **2** is file server **206**. In many LAN environments various server systems are provided which are shared resources for use and access by the users on the LAN. The file server **206** provides a shared disk storage resource for the LAN Administrator workstation **201** and the User workstations **202–204**.

Implementation Architecture of the Currently Preferred Embodiment

In the currently preferred embodiment of the present invention, an installation package is generated at the LAN administrator's workstation that subsequently causes installation of the software on each of the target workstations. Each of the target workstation contains means for receiving the installation package and processing it so that the application software is installed. It should further be noted that the present invention assumes that the Operating System and Operating System levels and other systems files on the LAN administrator workstation and the various user workstations, are the same. If they were not the same, it is clear that the potential for error would be great.

FIG. **3** illustrates the architecture of the operating software residing on the LAN administrator workstation and the target user workstation in greater detail. Referring to FIG. **3**, a LAN administrator workstation **201** contains LAN administrators tools **301**. The LAN administrators tools **301** include an installation scheduler **301***a* and installation package generator **301***b*. The installation scheduler **301***a* will allow the LAN administrator to schedule the automated installation of the new application software onto one or more user workstations. It is important to note that the installation scheduler **301***a* allows the LAN administrator to perform the actual installation at a time when the LAN administrator may or may not be physically present. The installation package generator **301***b* is used to create the installation package by which the application software is installed. In the currently preferred embodiment, installation package **303** is stored on file server **206**.

The target system i.e. the user workstations, contains a set of LAN client administration tools **302**. The LAN client administration tools **302** on the target user workstation **202** include a termination and stay resident (TSR) program **302***a* which remains active in memory on the target workstation. The remote program resident on the LAN Administrator's workstation executes this TSR program and causes an installation program (here installation program **302***b*) to be invoked. TSR programs of this type are well known to those skilled in the art, so further description of the TSR program is not deemed necessary. The installation program **302***b* is also part of LAN client administration tools **302**. The installation program **302***b* processes an installation package for installing the application software on the target system.

The manner in which this is done will be described in greater detail below. Finally, a deinstallation program **302***c* is provided. The deinstallation program **302***c* will process a deinstallation package for removing applications from the target system.

Installation Package

The installation package of the currently preferred embodiment of the present invention is generated in what is termed an IPACK format file. The IPACK format file defines various groups of commands which are used by the installation program residing on the user workstation to install the application software. Further, the UPACK format file provides a series of commands for deinstallation of files on a computer system. The IPACK format file is described in greater detail in Appendix A.

In the currently preferred embodiment, the installation package includes the IPACK format file and the application software to be installed. This package is stored on a server system in a compressed format, and it will be later downloaded during the installation process. As is described in Appendix A, the IPACK format file will contain references to the actual physical location of the installation package in compressed format.

Personality File

A second file utilized in the present invention is the personality file. The personality file allows for custom installation of application software on a user workstation. For example, if the application is to be installed in a particular directory, it is specified through the personality file. A custom personality file resides on the user workstation. In operation, the installation program on the user workstation will search for a custom personality file. If no custom personality file is found, a default personality file will be utilized to perform the installation. The personality file format is described in greater detail in appendix B.

The incorporation of a personality file provides a great deal of flexibility for the installation of application software. A custom personality file would typically be created by the LAN Administrator according to the unique requirements of the user.

Remote Installation of Application Software

Remote installation of Application Software by the LAN Administrator is a two-step process; generation of an installation package and scheduling the installation at each of the desired user workstations. FIG. **4** is a flowchart which illustrates the steps performed by the LAN administrator workstation and the user workstations. In this description, the LAN administrator workstation will be referred to as the source workstation and the user workstations will be referred to as the target workstations. First, on the source workstation a preinstallation system snapshot is generated and saved, step **401**. This preinstallation system snapshot will contain all the information that may be changed as a result of the installation of the application software, such as the complete disk directory structure contents, copies of various system and execution files (e.g. the CONFIG.SYS file and all *.INI and *.BAT files) and other operating system related information. Once the pre-installation system snapshot is stored the application software is installed, step **402**. The installation of the application software is performed according to the instructions provided by the particular software application. In the DOS and WINDOWS environment of the currently preferred embodiment, the installation procedures for application software is not standardized. Thus, the present invention has been designed to be general enough to allow for various application installation scenarios. In any event, after the application software has been installed on the source workstation, a post installation system snapshot is generated, step **403**. The post installation system snapshot contains the same information as the preinstallation system snapshot except that is taken after the application has been installed.

An installation package is then generated by comparing the preinstallation system snapshot with the post installation snapshot, step **404**. As described above, the installation package will include the IPACK format file and the application software. The generation of the installation package is further illustrated in FIG. **5**. FIG. **5** is a block diagram which illustrates the inputs and the resultant output for generating an installation package. The program IPACKGEN **503** takes as inputs the pre-installation snapshot **501** and the post installation snapshot **502**. As noted previously the preinstallation snapshot contains the state of the system prior to the installation of the application software. The post installation snapshot **502** contains the same information except that it is after the installation of the application software. IPACKGEN **503** then compares the pre-installation snapshot **501** with the post installation snapshot **502** to determine their differences. When comparing the preinstallation snapshot with the post-installation snapshot, the respective disk directory structures, CONFIG.SYS, *.INI, *.BAT and other system related files are compared. The IPACK format file contains instructions to enable modification of the preinstallation snapshot so that it equals the post-installation files. The installation package **504** is then generated based on these differences (refer to Appendix A for the specific commands used and for various examples). In the currently preferred embodiment, the installation package is stored on a server system on the LAN.

Referring back to FIG. **4**, once the installation package is created the installation of the application software on each of the respective targets systems is scheduled, step **405**. Such scheduling allows the LAN administrator to cause the installation of the application software at off hours, e.g. at night. It should be noted that in the currently preferred embodiment, the various steps performed by the LAN administrator are done via a dialog and windows type interface. The LAN administrator would respond to prompts from the dialog that has been created for generating installation packages and the scheduling of installations.

The mechanism for causing the transmission of the installation package to a target user workstation utilizes the system clock as a trigger. Such a mechanism for triggering the transmission utilizing the system clock is a well known technique to those skilled in the art. Thus further description of such a technique is not deemed necessary.

Assuming that all installations successful, the application software may optionally be deleted from the source system, step **406**. This is provided for because most application software licenses are based on the number of workstations which may use the software. If the LAN administrator is not going to be using that particular application software they would not want to count that copy of the application software on the LAN administrator systems towards the number being used.

The operation of the target workstation is now described. Referring back to FIG. **4**, at a scheduled time, the remote program on the LAN Administrator's workstation executes the TSR. At this point the install software will be invoked on the target workstation and will save the necessary files before the installation takes place, step **407**. These files are saved so that in the event of an error, e.g. network failure

during transmission of the application software, the application can be readily deinstalled. In any event, the installation software then looks to see whether a custom personality file exists, step **408**. If a custom personality file does not exist the application is installed based on the default personality file, step **409** otherwise, the application software is installed based on the custom personality file, step **410**. The operations occurring on the target side require no user interaction. The installation package contains the information needed to cause the application software to be installed.

The installation of the application software on the target workstation is further described with reference to FIG. **6**. FIG. **6** is a block diagram which shows the installation of the application software on a target system. An installer (or installation program) **601** receives as input installation packet **602** and a personality file **603**. Using these two files as input, an installed application **604** is created on the target system. This is accomplished by carrying out the installation commands set forth by the PACK format file in accordance with parameters set out by the personality file. As noted previously, the installer **601** also saves the system and data files before the actual installation **605**. As noted previously, the installer is initiated by a TSR program that is resident in the target system.

The installation program processes the PACK format file and determines what files are needed and where they are placed. It then obtains these files from the compressed installation package on the server and places them in the appropriate directories. It also modifies system configuration files based on the directives provided in the IPACK format file. For example in the case of Windows applications, it creates a group (if necessary) and icons as dictated by the application software.

Deinstallation of Application Software

Mechanically, the deinstallation of application software is substantially similar to those for the installation of application software. Here, a UPACK format file for the deinstallation of application software is generated. As a system snapshot has been created on each of the target workstations, prior to installation of the application software, this version of the various files are restored. A UPACK format file can be generally considered as the reverse of an IPACK format file, where added files are deleted. The specific instructions in the UPACK format file are described in Appendix A.

Thus, a method for installation of application software from a LAN Administrator workstation on to a user workstation on a network is described.

APPENDIX A

DESCRIPTION OF IPACK FORMAT FILE

The IPack Format File

The IPack format file is a text file that is created by a Windows utility program called IPackGen, that executes on your (the LAN administrator's) workstation after an installation package is created. The IPack file's input files are created by a program called Snapshot before the software is installed locally (during the creation of an installation package). These input files contain the following:

- Complete disk directory structure and contents (with file names, date-and-time stamps, file size, and so on)
- Saved copies of CONFIG.SYS, and all *.INI and *.BAT files.
- All Windows Shell (like Program Manager) group names and complete contents.
- Complete Windows Registration Database information.

To build the IPack format file, the IPackGen program compares the input files gathered by the Snapshot program with the similar information gathered by IPackGen after the software is installed. Using the differences, IPackGen creates the IPack format file.

LANInstall may also create a UPack, or Uninstall file for removing installations that have been made previously by LANInstall.

Appendix A ..e IPack Format File

General Format

The IPack Format file consists of 11 groups, presented in no particular order. Each group is denoted by a name enclosed in double square brackets. Some of these groups may be empty and are listed below in alphabetical order:

[[AddedFiles]]

[[AddedRegDB]]

[[BasePersonality]]

[[Configuration]]

[[DeletedFiles]]

[[DeletedRegDB]]

[[MergedPersonality]]

[[ModifiedTextFiles]]

[[ModifiedWinShell]]

[[OtherErrorsAndWarnings]]

[[ReplacedFiles]]

note icon *Although the names are given in mixed case, both case and leading whitespace are ignored. (For example," [[ADDEDFiles]]" is acceptable.) However,all characters inside the brackets are significant, and must match exactly.*

Any line whose first character is a semi-colon (";") is considered a comment and is ignored. Commented lines are copied (for example, from IPack to Log, from Personality to IPack, from Log to UPack), with the following exceptions:

- All comments are removed from **[[OtherErrorsAndWarnings]]**, **[[AddedFiles]]**, **[[DeletedFiles]]**, and **[[ReplacedFiles]]** groups.
- Comments in **[[AddedRegDB]]** are copied into **[[DeletedRegDB]]** (and vice-versa) when UPGen creates an uninstall IPack Format file (for instance, a FilterUPack or LogUPack file).

Appendix A The IP. . Format File

- Comments in the [[MergedPersonality]] group (which is created by merging the Base Personality with a workstation Personality) will contain only those comments from the file that is being overwritten (for instance, acting as "base").. That is:
  - Comments in the [InstallFlags], [UninstallFlags], or [FileTransferFlags] section will be copied only from the Base Personality (since the workstation Personality "Flags" sections always overwrites the Base Personality "Flags" sections).
  - Comments in the [PathMacros] section will be copied only from whichever Personality's [PathMacros] section is overwritten (as determined by the BaseMacrosOverwrite flag).

IPack Format File Groups

The next several sections explain the groups that make up the IPack format file.

[[AddedFiles]], [[DeletedFiles]], [[ReplacedFiles]]

The[[AddedFiles]], [[DeletedFiles]], and [[ReplacedFiles]] groups contain lists of directories and filenames. The format consists of a full directory path, followed by filenames within that directory. Sometimes a group may be empty. Additional sets of directories and filenames can be given as needed.

Following is a sample [[AddedFiles]] group:

```
[[AddedFiles]]
$(WINDOWS)\W4W
    DECOMP.EXE
    SETUP.EXE
        .
        .
        .
$(WINDOWS)\W4W\CLIPART
    BOOKS.WMF
```

Appendix A ...e IPack Format File

```
    CAPITOL.WMF
        .
        .
        .
$(WINDOWS)
    WINWORD.INI
```

Entries in the **[[ReplacedFiles]]** group are only valid within a Log file (see Appendix C for further discussion). Entries in this group are ignored during an installation. The IPackGen utility always places added and replaced files in the **[[AddedFiles]]** group. The Administrator's Log file tells whether a file was added or replaced during the initial installation that is part of creating an installation package.

IPackGen creates these groups in the IPack Format and Log files by comparing the data on all files and directories that are scanned before and after the initial installation. Any file or directory that changed (added, deleted, or replaced) is included in the appropriate group. The only exceptions to this are:

- The temporary working directory created by Snapshot (which is removed by IPackGen) is never included, nor are any files within that directory.
- No files that have both hidden and system attributes are included.
- Windows *.GRP files are not included (except for .GRP files that get added without a corresponding Windows Shell group).

During the installation, a check of the time and date stamps ensures that an existing file is replaced only if the added file is newer than the existing file. If it is not, an entry in the log file notes that the file was *not* replaced (unless the two files are identical). A Personality file flag, **OverwriteNewerFiles**, can be set to True to disable the time and date check, which forces the added file to be copied.

The **[[DeletedFiles]]** group is always processed first during an installation, followed by the **[[AddedFiles]]** group. Although unlikely, if a file appears in both lists, the existing file is deleted (if there is one) and the new file is added.

[[AddedFiles]]

Appendix A The IP. Format File

During an installation, if a directory listed in the [[AddedFiles]] group does not exist, it is created.

In one type of IPack Format file, a FilterUPack, a "*" may appear after a directory name. A FilterUpack file is sometimes part of the deinstallation process. for more information, see "Uninstall IPack Files" in appendix D. The trailing * indicates that all files listed under this directory in the LogUPack file being filtered are acceptable and may be added.

[[DeletedFiles]]

In the [[DeletedFiles]] group, a directory name followed by a list of filenames indicates that the listed files are to be deleted and that the directory (and any unnamed files) will remain. If no filenames are specified, no action will occur.

A directory name ending with * indicates that the directory and all of its files will be deleted. (This includes all sub-directories and *their* files.) Any filenames following the directory name are ignored.

A directory name ending with \? indicates that the directory will be deleted only if it is empty after deleting all the other files and sub-directories listed in the [[DeletedFiles]] group.

[[AddedRegDB]] and [[DeletedRegDB]]

The [[AddedRegDB]] and [[DeletedRegDB]] groups are non-empty only for installations of Windows programs that modify the Registration Database. If one or both of these groups contain entries, the Windows macro must be defined (see Appendix B for details).

[[AddedRegDB]]

Entries in the [[AddedRegDB]] group have the following form:

key=value

key refers to the name of the key as stored in the Registration Database. It is similar to a directory path, but does not begin with the root indicator (\).

value is the string to the right of the equal sign, and may consist of any characters.

For example:

Appendix A : IPack Format File

```
regedit\shell\open\command=regedit.exe %1
```

During an installation, keys listed in this group are added to the workstation's Registration Database if they do not already exist. If the key already exists, but has a different value than the value specified, the value is changed to the specified value. However, an existing value is *not* be replaced with a null value. (A null value is given by an entry of the form "key=", that is, no value is specified to the right of the equal sign.)

[[DeletedRegDB]]

Entries in the [[DeletedRegDB]] group have the following form:

key

Only *key* names are listed, since the value is not considered when deleting a key. (The Log file *does* contain the value, however, as described in Appendix C).

The keys given in this group are deleted, as are sub-keys branching from each specified key. For example, assume the Registration Database at a particular workstation contains the following keys:

```
key0
key0\key1
key0\key1\key2
key0\key1\key2\key3
key0\key1\keyTwo
key1
```

If `key0\key1` is listed in the [[DeleteRegDB]] group, the following keys are deleted:

```
key0\key1
key0\key1\key2
key0\key1\key2\key3
key0\key1\keyTwo
```

[[BasePersonality]]

The [[BasePersonality]] group usually contains a copy of the default Base Personality file present on the LAN admin's workstation during the creation of the IPack format file.

note icon *If the personality file specified in Parmfile.ss is not the default personality file on the administrator's workstation, the file copied is the one specified in the file Parmfile.ss.*

Appendix A The IP. Format File

The [PathMacros] section is copied as read. Rather than copying all of the flags sections in the personality file, only the appropriate flags sections get copied (For a file transfer, this would be the [FileTransferFlags] section. The [UninstallFlags] section is copied also.)

During installation, this information is used only if a search of the workstation's Personality file fails to find a required path or flag entry. For more information on these entries, refer to Appendix B, *Personality Files*.

The only time the [[BasePersonality]] group contains information other than that read from the base Personality file is during the deinstallation process, within a UPack, LogUPack, FilterUPack, or a Log file. In these cases, the group's information still represents the base (default) Personality. However, the information is derived from sources other than the base Personality file. This is necessary to properly "undo" an installation where the Personality file has changed since the installation was performed.

[[Configuration]]

Thie [[Configuration]] group is always required to be non-empty and contains entries that pertain to the configuration of the package being installed. These entries are explained as follows:

ExcludePath=*Path*

The `ExcludePath=` entry specifies a path to a directory that contains files to be excluded from the installation. It must be created by hand because it is never created by the IPackGen utility. If the `ExcludeFiles` flag in a workstation's Personality file is set to True, no files from the specified directory are copied, nor is the directory created. (This also includes creation or modification of text files given in the `[[ModifiedTextFiles]]` group.) All directories that branch from the specified directory are also skipped.

The `ExcludePath=` entry may be given more than once if it is necessary to specify more than one directory path.

IPackDataFile=*Path\Filename*

Appendix A , IPack Format File

The `IPackDataFile=` entry specifies the complete path and filename of the compressed "archival" file that contains copies of all files that must be copied during installation of the software package. This entry is created by IPackGen.

For more information, see Appendix C, *Log Files*.

IPackFileType=*Type*

The `IPackFileType=` entry identifies the type of the IPack Format file and is created by IPackGen (or DPackGen or UPGen, as appropriate). The `IPackFileType=` enty must contain one of the following values:

`IPack` An an installation IPack Format file.

`DPack` A file transfer (distribution) IPack Format file.

`UPack` A "merged" de-installation IPack Format file, created by merging a FilterUPack and LogUPack file.

`FilterUPack`
A file used to filter entries in a LogUPack file during the deinstallation process. "Filter" means to modify an event or prevent it from happening. FilterUPack is created from an IPack file during the deinstallation process.

`LogUPack` A UPack Format file created from a Log file during the deinstallation process.

`Log` A log file (created during installation, deinstallation, or file transfer).

`Error` One or more errors were encountered during generation of this file.

`ErrorFatal`
A fatal error was encountered, forcing processing to terminate prior to completion.

RequirementsFile=*Path\Filename*

The `RequirementsFile=` entry specifies the complete path and filename of the file that contains the hardware and software requirements for the software package being installed. IPackGen creates the `RequirementsFile=` entry by copying the corresponding entry given in the Parmfile.li file.

Appendix A The IP. Format File

[[MergedPersonality]]

The [[MergedPersonality]] group contains a copy of the Personality file that was used during creation of this IPack Format file. It contains the macros and flags that resulted from the merging of the base Personality and the LAN Administrator's Personality.

This information is used by DPackZip, and by UPackGen when processing an IPack Format file directly (not a Log file). For more information, refer to Appendix B, *Personality Files*.

[[ModifiedTextFiles]]

The [[ModifiedTextFiles]] group uses a script language to detail the changes that must be made to specific text files during the installation process. Text files that may be modified include *.BAT, CONFIG.SYS, *.INI, and possibly other files as determined by the creator of the IPack Format file (such files will be given a file type of INI).

note icon *Changes made to the "Order=" entry in the [Settings] section of the "PROGMAN.INI" file, as well as to the entire [Groups] section, should never be given in this group. Such changes must be made by using the appropriate commands available in the [[ModifiedWinShell]] group detailed later in this appendix.*

The [[ModifiedTextFiles]] group is created by IPackGen when it encounters a text file that has been added or changed. A deleted text file, however, is not entered in this group (instead, it appears in the [[DeletedFiles]] group). This does not prevent the deletion of a text file from occurring in this group. For example, a text file that was created during a LANInstall installation is "uninstalled" by a series of Remove commands. If the text file is empty after processing all the commands, it is deleted.

Notation Conventions and Rules

The following notation conventions pertain to the commands that may be included in the [[ModifiedTextFiles]] group:

Appendix A e IPack Format File

- Text in *italics* denotes a replaceable parameter. It must be replaced with suitable information (see below). ***Bold*** text is used to denote keywords (like commands).
- Replaceable parameters shown in braces (like "{*Where*}") are optional.
- The OR character (|) between two elements indicates "one or the other, but not both."
- All text is case insensitive.
- Leading whitespace is ignored in most instances. However, whitespace is necessary to delimit tokens, except where a distinct delimiter exists (like a bracket ([]).
- A line may contain up to 510 characters. A command cannot continue on to the next line. A line whose first non-whitespace character is a semicolon is considered a comment and is ignored.
- If a command matches more than one line within a file (or within a section, if dealing with sections), the first line matched will be the line acted upon.

Replaceable Parameters

FindText

Locates a line in the original file by combining 1 or more {*Text*} search strings. For example, {{rem} {device} {mouse.sys}} will locate a line containing all three strings (in the order given).

FindText must be surrounded by parentheses. Consider the following example:

```
Append ({path}{=}) {$(WINDOWS)\ACCESS}
```

The search for a matching line containing *FindText* always begins at the top of the original file (or section, if a section is being modified). *FindText* locates the first matching line in the original file that remains unmodified in the new (working) file. In other words, if a line containing *FindText* is found, but that line has been **Delete**'d, **Rem**'d, or **Unrem**'d in the working file, the line will not be used, and the a search for the next matching line begins.

A line will match *FindText* only if the *FindText* strings occur in the line in the same order as given in *FindText*. For example, {key}{=} requires that the equal sign character occurs after the matched "key"string.

FindText matches are performed in the same manner as a search for a line to be added (see ***Add matches*** later in this chapter). Enough *FindText* must be given for at least a partial match; more information can be given to ensure an exact match. For example, given an AUTOEXEC.BAT file containing:

```
set var1 = i:
```

the *FindText*:

```
({set}{var1})
```

will match with the AUTOEXEC.BAT line. A more specific *FindText*, such as:

```
({set}{var1}{C:})
```

will not match this line, since `C:` represents the value portion of the line, and the existing line has a value of `j:`, not `C:`.

In order to match comment lines, the proper comment character(s) must be given as the first *FindText* item; comment lines will not match *FindText* that does not include the comment `path = c:\dos;c\windows`

The *FindText* `({path}{=})` only matches the second line, not the first.

Path/Filename

A full path, including drive letter and colon, followed by a filename (including extension, if any). May include one or more path macros (for example `$(DOSDRIVE)\FILE.EXT`).

Section

For files containing sections (like .INI files), `[Section]` must be specified in the first command following a **File** command; it is optional in subsequent commands applying to the same section. If the section doesn't exist in the original file, all commands except **Add** and **Append** will log an error and skip the command. **Add** and **Append** will create a new section if `[Section]` does not exist.

Appendix A : IPack Format File

If the `[Section]` specified is commented out, the comment character(s) must appear as the first character of `[Section]` (for example, `;[fonts]`).

Comment lines at the beginning of an .INI file, before any `[Section]`, are referenced by using the special "null section" notation: "`[]`" (no characters between the open and close brackets).

The search for a matching section always begins at the top of the original file. Except for **Add** and **Append**, the matching section must also exist in the current file (the one being modified). If the section is in the original file, only a **Delete** command will remove it from the current file (or **Remove**, if the **RemoveIsDelete** flag is True).

Text

A complete or partial statement within a text file (like SYSTEM.INI, AUTOEXEC.BAT, and so on). It may include replaceable parameters, like `$(DOSDRIVE)`, anywhere within the string.

Text must always be enclosed within braces. For example,

```
{This is a Text string.}
```

If *Text* contains braces, they must properly matched. The following is invalid:

```
{This is an open-brace: "{" but there is
no matching end-brace.}
```

Where

Where identifies the position in the text file the specified line is to be added. If *Where* is not given, the line will be added after the previously processed line. If no line has been processed yet in the file being modified, the line will be added to the end of the file. *Where* is not applicable to Windows .INI files, since they are not order-dependent. Additions made to these files are always made at the end of the current section (and sections will always be added to the end of the file being modified).

Where is replaced by one of the following commands:

After *FindText*	The line below the first line matching *FindText*.
Before *FindText*	The line above the first line matching *FindText*.

End	The last line of the file.
Next	The line after the previous line just processed (default if *Where* is not specified).
Start	The first line of the file.

As currently implemented, Next's "line just processed" includes a line processed by *any* command, not just Add. This means that if lines are being added to the end of a file, and then a line is Rem'd, the next line will be added after the Rem'd line (unless a *Where* clause is given).

$(*MacroName*)
A macro name to be replaced with a path during the actual installation (based on the macro's value in the Personality file on the workstation where the installation is taking place).

[[ModifiedTextFiles]] Commands

This section details the commands used in the [[ModifiedTextFiles]] group.

File *Path\Filename*

The File command denotes that any following commands are to be applied to the specified file. This command is required, and must appear as the first statement for each file to be modified. *Filename* must be CONFIG.SYS, *.BAT, or *.INI. Any other filename will be processed as if it is an .INI file, which may lead to errors.

The following commands appear after a File command, and apply to the specified file:

Add {[*Section*]} {*Where*} *Text*
This command adds the new line *Text* to the current file being modified. Add must be replaced by one of the following:

Add!	Adds the new line even if an identical line already exists in the current file.
Add+	Adds the new line as long as an identical line does not already exist.

Add Adds the new line if no similar line exists, otherwise the existing line will be **Remove**'d ("translating" to either **Delete** or **Remout**) and the new line added.

Add? Adds the new line only if no similar line exists.

Three of the four forms of the **Add** command add a new line without disturbing existing lines. The fourth form, "**Add** " may add a new line, or replace an existing line. When searching for an existing similar line, the entire original file is searched (before changes). Just as with *FindText* searches, only matching lines that are unmodified in the new (working) file are used. If a *Where* clause designating a specific location is given (**before** or **after** a specific line), then only that specific line will be examined for a match with the line being added—the entire file will not be searched. (This ensures that the line is added where specified, regardless of whether the line is new, or it replaces an existing line.) If one of the other *Where* clause locations is given (**end**, **next**, or **start**), the entire file is searched. In this case, the *Where* location will be used only if the line is new.

For a description of matching "identical" and "similar" line, see "**Add** matches" later in this chapter.

Add creates a new section if *Section* does not already exist. An empty section is be created if only "**Add** [*Section*]" is given (and *Text* is not given). (However, even the command "**Add!** [section]" will not add [*Section*] if it already exists.)

This command has the opposite effect of the **Delete** command (that is, a **Delete** command in an Install Log file will be translated to a corresponding **Add** command for inclusion in a UPack format file).

Append {[*Section*]} *FindText Text*

Appends *Text* to the end of the first statement found that matches *FindText* within the current file (or section). The first character of *Text* is assumed to be the delimiter between items being appended. The installation routine ensures that two consecutive delimiters are not placed together. For example, consider the following command line:

```
Append ((path)) (;c:\windows;d:\windows\winword)
```

If the file being modified has a path statement ending with a semicolon, the first semicolon in the **Append** command line will not be added.

Multiple items may be appended by including them within *Text*, separated by the append delimiter. If an item already exists in the line matching *FindText*, it will not be appended. (The comparison for an existing item is case insensitive.) The following example appends only `calendar.exe` to the line `load=clock.exe calc.exe cardfile.exe`:

```
Append ({load}) {calendar.exe CALC.EXE Cardfile.exe}
```

If the **AppendAndExtractSaveOriginal** flag in the Personality file is set to True, a commented-out copy of the original (unmodified) line is **Add**'d to the file being modified.

If no line matching *FindText* is found, a line created from the **Append** command line is added to the end of the file (or section). The line is created by concatenating the *FindText* elements, each separated by a space, followed by an equal sign character, and *Text* (without the first [delimiter] character). Consider the following example:

```
Append ({set}{temp}) {;$(WINDEV)}
```

It creates the following line if no line containing "set" and "temp" is found:

```
set temp=$(WINDEV)
```

Although the line was added, it appears in the Log file as an **Append** operation.

This command is used as the opposite of the **Extract** command (that is, an **Extract** command in an Install Log file will be translated to a corresponding **Append** command for inclusion in a UPack format file).

Delete {[*Section*]{?|*}} *FindText*

The **Delete** command deletes the first line encountered containing *FindText* within the file (or section). The original file's contents will be searched (as opposed to the current file after modifications).

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified will be searched. (If a *Section* has not yet been specified for this file, no delete will occur.)

Appendix A : IPack Format File

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be deleted, including all entries within that section.

A '?' given after [*Section*] specifies that the section be deleted only if it is empty (i.e., contains no entries, including comments, but not blank lines). If it is followed by *FindText*, then the specified line will be deleted before the check for an empty section is made.

Delete [*Section*] (without a '?', '*', or *FindText*) is invalid and will be ignored.

Extract {[*Section*]} *FindText Text*

Removes *Text* from the first statement found matching *FindText* within the current file (or section). The **Extract** operation fails if a statement matching *FindText* is not found.

The first character of *Text* must be the "append delimiter" between items being extracted. Multiple items may be extracted by including them within *Text*, separated by the "append delimiter." The comparison for an existing item is case insensitive. Consider the following example:

```
Extract {{load}} { calendar.exe CALC.EXE Cardfile.exe}
```

It extracts `cardfile.exe` and `calc.exe` from the following line:

```
load=clock.exe calc.exe cardfile.exe
```

If the **AppendAndExtractSaveOriginal** flag in the Personality file is set to True, a commented-out copy of the original (unmodified) line will be **Add**'d to the file being modified.

The **Extract** command is used as the opposite of the **Append** command (that is, an **Append** command in an Install Log file will be translated to a corresponding **Extract** command for inclusion in a UPack format file). The **Extract** command allows text that was previously added using **Append** to be removed from within the same line, no matter where it currently is within that line.

Remout {[*Section*]{?|*}} *FindText*

Creates a remark from the first line encountered containing *FindText* within the file (or section). (If the line is already commented out, no action will occur.) Remarks are preceded by "rem" unless the file is an .INI file, in which case a semicolon appears.

Appendix A The II- . Format File

If [*Section*] is not given, but the file being modified contains sections, the last *Section* specified is searched. (If a *Section* has not yet been specified for this file, no action will occur.)

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be commented out, including all entries within that section.

A '?' given after [*Section*] specifies that the section heading (name) be commented out, if the section is empty (commented and blank lines are not considered in determining whether the section is empty or not).

Remout [*Section*] (without a '?', '*', or *FindText*) comments out the section heading only.

Remout is used as the opposite of the **Unrem** command.

Remove {[*Section*]{? | *}} *FindText*

The **Remove** command removes the specified line by acting as either a **Delete** or **Remout** command. By default, **Remove** acts as a **Remout** command, commenting out the first line encountered containing *FindText* within the file (or section). If the line is already commented out, no action will occurs. The original file's contents will be searched, as opposed to the current file after modifications. However, if the **RemoveIsDelete** flag in the Personality file is set to True, then **Remove** will actually **Delete** the specified line.

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified is searched for a *FindText* match. If a *Section* has not yet been specified for this file, no remove occurs.

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section will be removed, including all entries within that section.

A '?' given after [*Section*] specifies that the section be removed only if it is empty (contains no non-blank entries, or comments if the **RemoveIsDelete** flag is True). If it is followed by *FindText*, then the specified line will be removed before the check for an empty section is made.

Remove never appears in the Log file; it is always changed to **Delete** or **Remout**, depending on which action really occurred. Defining **Remove** this way allows the LAN Administrator to select a default action for **Remove** and yet manually override specific instances by editing the IPack Format file. For example, if the **RemoveIsDelete** flag is set to False, but you know that the command **Remove** *(Files=40)* can safely delete the line "Files=40", you can change the command to **Delete**.)

Remove is used as the opposite of the **Add** command (that is, an **Add** command in an Install Log file will be translated to a corresponding **Remove** command for inclusion in a UPack format file). Since **Remove** itself never appears in a Log file, it has no opposite command.

Unrem {[*Section*]}{*} *FindText*

Removes a remark from the first line encountered containing *FindText* within the file (or section). The remark characters removed will be "rem" unless the file is an .INI file, in which case ";" will be removed. As mentioned in the *FindText* discussion in "Replaceable Parameters" earlier in this appendix, the remark character(s) must be given as the first *FindText* item in order to find the line to **Unrem**.

If the specified [*Section*] refers to a section heading (name) that is commented out in the original file, the comment character(s) must be given as the first character (for example, in an .INI file `; [Colors]`).

If [*Section*] is not given but the file being modified contains sections, the last *Section* specified will be searched. If a *Section* has not yet been specified for this file, no action occurs.

A '*' given after [*Section*] name (without *FindText*) specifies that the entire section be uncommented, including all entries within that section.

Unrem [*Section*] (without a '*' or *FindText*), uncomments the section heading only.

Unrem is used as the opposite of the **Remout** command

[[ModifiedTextFiles]] Examples

Appendix A The II Format File

Consider the following [[ModifiedTextFiles]] section of an IPack format file:

```
[[ModifiedTextFiles]]
FILE $(DOS)\AUTOEXEC.BAT
APPEND ((path)(=)) ($(WINDOWS)\ACCESS)
ADD NEXT (SET INIT=$(WINDOWS)\ACCESS)
ADD NEXT (SET TEMP=$(WINDOWS)\ACCESS\TEMP)
FILE $(WINDOWS)\SYSTEM.INI
ADD [386Enh] (device=*vmcpd)
REMOVE ((device)(=)(vmcpd))
```

The preceding commands would make the following changes to the AUTOEXEC.BAT file found in the $(DOS) directory.:

- The **Append** command adds the base path to the end of the current "path" variable found in AUTOEXEC.BAT. If the flag **AppendAndExtractSaveOriginal** in the Personality file is set to True, a commented-out copy of the original `path=` statement is added.
- The line `(SET INIT=$(WINDOWS)\ACCESS)` is added after the `path=` statement. If a `set init=` statement already exists in this position within the file, it is **Remove**'d. If the flag **RemoveIsDelete** in the Personality file is set to False (or is not defined), then a **Remout** of the current `set init=` statement is performed, otherwise a **Delete** occurs.
- The line `SET TEMP= $(WINDOWS)\ACCESS\TEMP)` is added after the `set init=` statement. A **Remove** of an existing similar line occurs as just described.
- The line `device=*vmcpd` is added to the end of the `[386Enh]` section of SYSTEM.INI. If a line containing `device=` and vmcpd already exists in this section, it is **Remove**'d.
- The **Remove** command operates on the "current" section (`[386Enh]`). It affects the original `device=*vmcpd` statement, not the one added by the preceding command. If the flag **RemoveIsDelete** in the Personality file is set to True, the specified statement is deleted. If the **RemoveIsDelete** flag is set to False, a **Remout** of the statement occurs. This command is redundant because the existing similar line would have been **Remove**'d during the **Add** command described earlier.

Appendix A e IPack Format File

Assume CONFIG.SYS is being modified, and it contains the following line:

```
DEVICE=$(WINDOWS)\HIMEM.SYS
```

The following statement:

```
ADD (device=$(DOS)\HIMEM.SYS)
```

Replaces the line:

```
DEVICE=$(WINDOWS)\HIMEM.SYS
```

with:

```
device=$(DOS)\HIMEM.SYS
```

after making the appropriate substitutions of `$(WINDOWS)` and `$(DOS)`. The Log file will show[1]:

```
Delete (DEVICE=$(WINDOWS)\HIMEM.SYS)
Add (device=$(DOS)\HIMEM.SYS)
```

This example assumes that the flag **RemoveIsDelete** in the Personality file is set to True (if not, the **Delete** would be a **Remout**). and illustrates that an **Add** command performs a **Remove** on an existing line (if there is one). The example also assumes that paths are used in determining a match (see **Add matches**, later in this appendix). If they are not, then no change is made, since the root program name in the line to be added is the same as that of the existing line.

Here is another example:

```
Add [fonts]
Add (Small Fonts (VGA res)=SMALLE.FON)
```

The sequence adds the following line to the end of the [fonts] section in the current file:

```
Small Fonts (VGA res)=SMALLE.FON
```

If a section named [fonts] does not already exist, it is created. "Add [fonts]" is written to the Log file as "**Add** [fonts] +" if the section was created.

Add matches

[1]The actual macro name used in the **Add** statement log may be different from that given in the Ipack Format, since it depends upon the workstation's Personality file.

Appendix A The IP. Format File

The **Add** command searches the "original" copy of the current text file being modified for a line that matches the line being added, in order to determine what action to perform (replace the current matching line or just add the new line). It searches for a match in the original copy, and then references the matching line's corresponding line in the "working" copy.)

The range of lines that may be searched is limited as follows:

- If a *Where* clause designating a specific location is given (before or after a specific line), then only that specific line is examined for a match with the line being added
- If a *Where* clause is not given and the file contains sections, only the current section is searched.
- If a *Where* clause is not given and the file does not contain sections, all lines in the entire file are searched.

A match is either "identical" (also, "exact") or "similar" (also, "partial"). Whether a match is identical or similar is determined by the type of file being modified (CONFIG.SYS, *.BAT, or *.INI), as well as the specific line being added. Different line types have been defined, grouping together lines with similar characteristics. A line is composed of several elements; a specific line type has some or all of these elements. The exact meaning of the element may also depend upon the specific line type (or even the specific line).

The possible elements of a line are:

keyword
The keyword might be the command in a *.BAT file (for example, "ECHO"), or the key in an *.INI file (for example, "speed" in the line `speed=medium`).

key modifier
An additional piece of information required to make the keyword unique. Both the keyword and key modifier (when present) must be identical for a line to match. The key modifier is not applicable to *.INI files.

path
If a specified path is necessary to further qualify a line, it is stored in "path" (as opposed to being part of "value"). For example, the command `call path\progname` uses *path* as a unique part of the statement. *Path* is not applicable to *.INI files, which always use "value".

Appendix A ɜ IPack Format File

value

The remainder of the line, after the other elements have been identified. For example, "medium" in the *.INI line `speed=medium`, or `/e:1000 ./p` in the CONFIG.SYS command

```
shell=D:\msdos\command.com /e:1000 /p.
```

The different line types, the elements they include, and an example of each type are summarized in the following table.

Line Type Name	Keyword	Key Modifier	Path	Value	Example
Blank					Blank lines: always exact matches
BatStd		Partial	Exact	Exact	C:\Windows\smartdrv C 2048
CfgDevice	Exact	Partial	Exact	Exact	Device=C:\DOS\setver.exe C:
IniStd	Partial			Exact	speed=medium
KeyAndKeyMod	Partial	Partial		Exact	SET VAR1 = 2000
KeyAndProdName	Partial	Partial	Exact	Exact	Call D:\bats\dos_path DOS_MSC
KeyAndValue	Partial			Exact	Echo This is a test

Partial	Indicates that the element must match between the lines being compared in order for the lines to be considered at least "partial" matches.
Exact	Indicates that the element must match in order for the lines to be considered "exact" matches.

The following table shows how to determine the line type of a line based on the type of file the line is in, and the key word found in the line. It also lists the default Add command that IPackGen generates (for example, Add! is generated when adding a blank line).

File Type	Keyword	Line Type	Add Type
All files	<none>	Blank	!
CONFIG.SYS	Device	CfgDevice	
	Devicehigh	CfgDevice	
	Drivparm	KeyAndKeyMod	
	Install	KeyAndProgName	
	Shell	KeyAndProgName	

Appendix A The Il . Format File

	<all others>	KeyAndValue	
*.BAT files	Break	KeyAndValue	
	Call	KeyAndProgName	
	Cls	KeyAndValue	!
	Command	BatStd	?
	Echo	KeyAndValue	!
	Emm386	BatStd	?
	Fastopen	BatStd	?
	Lastdrive	KeyAndValue	
	Goto	KeyAndValue	!
	If	KeyAndValue	!
	Loadhigh	KeyAndProgName	
	Lh	KeyAndProgName	
	Mirror	BatStd	?
	Path	KeyAndValue	
	Panse	KeyAndValue	!
	Set	KeyAndKeyMod	
	Share	BatStd	
	Shift	KeyAndValue	!
	Verify	KeyAndValue	
	<all others>	BatStd	+
*.BAT files	Device	KeyAndProgName	
	<all others>	IniStd	

The "@" character (as the first non-whitespace character of a *.BAT line) is not considered significant when searching for matching lines. For example, the line "echo Hello!" being added with the command:

```
Add(echo Hello!)
```

will match exactly with the following line in a *.BAT file:

```
@echo Hello!
```

Comment lines only match other, identical comment lines. For example, the line "files = 50" with the command:

```
Add(files = 50)
```

will not even partially match with the following line in a CONFIG.SYS file:

```
rem files=50
```

The lines "rem files = 50" and "rem files=40" do not even partially match.

[[ModifiedWinShell]]

The [[ModifiedWinShell]] group will be non-empty only for installations of Windows programs that modify the Shell's group information. This group is one of the last two groups to be processed, after all other files have been copied and modified as needed.

Notation Conventions and Rules

The following notation conventions pertain to the commands that may be included in the [[ModifiedWinShell]] group:

- Text in braces ("{——}") is optional.
- Text in *italics* is replaced with suitable information (see below). **Bold** is used to denote keywords (like commands).
- All text is case insensitive.
- Leading whitespace between tokens (keywords) is ignored.
- A Command can not continue on to the next line. A command may contain up to 510 characters. A line whose first non-whitespace character is ";" is considered a comment and is ignored.

[[ModifiedWinShell]] Commands

The [[ModifiedWinShell]] commands modify group files. Quotation marks must be used to delimit string arguments that contain spaces, commas, or parentheses. The syntax and command names were adapted from the Windows' Shell-DDE interface.

AddItem(CmdLine,Name{,IConPath{,IconIndex{,XPos {,YPos{,DefDir{,HotKey{,fMinimize}}}}}}})

Appendix A The IF Format File

Adds an item (i.e., icon) to the currently active group. If an item already exists with the same *Name*, the existing item will be replaced by this item.

CmdLine
A string that specifies the full command line required to start the application. It can be the full path of an executable file along with parameters required by the application. It can also be an associated file name. The association is taken from the registration database.

Name
A string that specifies the item title to be displayed below the icon in the group window.

IconPath
A string that identifies the filename for the icon to be displayed in the group window. It can be either a Windows executable file or an icon file. If this parameter is not specified, Windows will attempt to use the *CmdLine* parameter. If *CmdLine* specifies neither an executable file nor an associated executable file, Windows will use a default icon.

IconIndex
An integer that specifies the index of the icon in the file identified by the *IconPath* parameter.

XPos,YPos
The *xPos* and *yPos* integers specify the horizontal and vertical positions of the icon in the group window. If these values are given as "-1", or no arguments are specified after *IconIndex*, the icon will be added in the next available position in the group window.

DefDir
A string that specifies the name of the default (or working) directory.

HotKey
A decimal integer that identifies a hot (or shortcut) key sequence specified for the application.

fMinimize
A Boolean flag (0 or 1) that specifies whether an application window should be minimized when it is first displayed.

CreateGroup(*GroupName*)

Creates a new group, or activates the window of an existing group. If the group exists, this command is written to the Log file as **ShowGroup** instead of **CreateGroup**. (**CreateGroup** appears in the log file only if the group was actually created.)

GroupName
A string that identifies the group to be created or activated.

DeleteGroup(*GroupName*){?}

Deletes an existing group, including all the items (icons) within the group. However, if a '?' is given after the closing parenthesis, the group is deleted only if it is empty at the time the command is processed.

GroupName A string that identifies the group to be deleted.

DeleteItem(*ItemName*)

Deletes the first item in the currently active group found with the name of *ItemName*. If deletions are the only actions to be made from the currently active group, activate **DeleteItem** it is with **ShowGroup** rather than **CreateGroup** (see **ShowGroup** later in this appendix).

ItemName
A string that specifies the item to be deleted from the currently active group.

ReplaceItem(*ItemName*)

Deletes the first item in the currently active group found with the name of *ItemName*, and remembers the position of this item (icon). The next command should be an **AddItem** command, which will be placed in the same position as the item just deleted, regardless of the **AddItem**'s *XPos* and *YPos* positions (if any are given).

ItemName
A string that specifies the item to be replaced in the currently active group.

ShowGroup(*GroupName*)

Appendix A The IF Format File

The ShowGroup command activates the specified group window. Unlike CreateGroup, the specified window is not created if it does not exist. ShowGroup should be used in place of CreateGroup when only deletions are being made from a group (to prevent creation of an empty window).

Appendix A ; IPack Format File

[[ModifiedWinShell]] Example

Here is an example of the [ModifiedWinShell]] group that creates a new group "Windows Applications", and adds a new program item (i.e., icon) "Win App" to this group.

```
[[ModifiedWinShell]]
CreateGroup("Windows Applications")
AddItem(winapp.exe,"Win App",winapp.exe,2)
```

[[OtherErrorsAndWarnings]]

The [[OtherErrorsAndWarnings]] group contains error and warning messages that do not directly relate to any other group.

APPENDIX B

DESCRIPTION OF PERSONALITY FILE

The Personality File

A LANInstall Personality file is a text file containing workstation-specific information that allows an installation to be customized for a particular workstation. On any given workstation, this Personality file is optional. A base, or default, Personality file is included in the IPack format file. For more information on this default file, see the [[BasePersonality]] group description in Appendix A.

A Personality file contains path macros and customization flags that are described in this chapter. If a workstation's Personality file does not contain a particular macro or flag, the value in the IPack format file's [[BasePersonality]] group is used. If the same entry is in both the workstation's Personality file and the [[BasePersonality]] group, the value of the **BasePersonalityOverwrites** flag determines which entry is used. For normal operation, this flag should be set to FALSE in the **[InstallFlags]** and **[FileTransferFlags]** sections of the Personality file, and TRUE in the **[UninstallFlags]** section.

Appendix B The Pe ality File

[PathMacros]

Specified path macros replace absolute paths, which allows you to customize installation parameters for individual workstations. They are defined in the Personality file section named [PathMacros], and have the form `MacroName=path`. Consider the following example:

```
[PathMacros]
UTILS=F:\UTILS
DOS = d:\msdos

WINDRIVE=G:
Windows=$(WINDRIVE)\windows
BOOTDIR=C:\
```

Case is ignored, but non-leading whitespace is significant. The Macro name on the left side of the equal sign must contain the same whitespace as found in the macro reference. For example, `$(WindowsDir)` matches `WindowsDir`, but not `Windows Dir`).

Path macros are entirely user-defined and are not created by LANInstall. Macro names may be anything you like, but there must be a macro named Windows defined if the installation involves a Windows program. The Windows macro must expand to the complete path of the directory in which Windows is installed. For example, `Windows=$(WINDRIVE)\windows`.

After final macro expansion, directory paths must be *full*, that is, they must always begin with a root directory, usually preceded by a drive letter and colon. As an example, say the Personality file contains the following line:

```
WINUTIL=J:\WINDOWS\UTILS
```

Now say that the (absolute) path of the installed package is:

```
J:\WINDOWS\UTILS\PBS
```

The resulting "macro-based" path would be:

```
$(WINUTIL)\PBS
```

If the macro-based path is set to an absolute path (such as `J:\WINDOWS\UTIL\PBS`), the software is installed in the same, absolute location on all workstations and warning message is written to the Log file.

Appendix B The Pe ıality File

A valid path does not end with a "\", except when referring to the root directory (like BOOTDIR=C:\) Then it is absolutely essential. If such a macro is defined without the "\" character, it refers to the specified drive, not the drive's root directory.

When possible, you should avoid defining multiple macros for the same path. For example, assume the Base Personality has a path macro defined as `Util = C:\util`. A workstation's Personality has `Util` and another (say, `UtilPath`) set to the same path. After an installation, the workstation's installation Log file may show `UtilPath` as the macro used, rather than `Util`. This is not likely to cause problems, but it certainly could cause confusion.

Appendix B The P. .ality File

[FileTransferFlags], [InstallFlags], [Uninstall Flags]

The [FileTransferFlags], [InstallFlags], and [UninstallFlags] sections of the Personality file contain Boolean flag values that can be included to personalize the procedure being performed (file transfer, installation, or uninstallation) for the workstation containing the Personality file. The following values are all acceptable (case is ignored):

True	False
T	F
Yes	No
Y	N
1	0

Default values are always False if the entry is not present, or a value is not given. Although the following flag names are shown in mixed case, case is not significant. Some flags may not be applicable to all sections. For example, **AppendAndExtractSaveOriginal** is ignored if it appears in the [FileTransferFlags] section.

AppendAndExtractSaveOriginal=*Bool*

If True, the **Append** and **Extract** commands of the [[ModifiedTextFiles]] group create (Add) a commented-out copy of the original line (before modification).

BackupOldVersion=*Bool*

If True, an archival copy containing all files that appear (uncommented) in the Log's [[DeletedFiles]] and [[ReplacedFiles]] groups is created, using the path and filename specified in the `iPackDataPath` parameter of the PARMFILE.LI file.

BaseMacrosOverwrite=*Bool*

If True, a macro in the `[[BasePersonality]]` group in the IPack format file takes precedence over the same macro in an individual workstation's Personality file. By default, if this entry is not present, the workstation's Personality file overwrites duplicate macros in the `[[BasePersonality]]` group.

For install and file transfer operations, this flag should normally be set to False. For an uninstall, this flag should normally be set to True. This ensures that the macros used are the macros that were in effect during the installation of the package that is being uninstalled. Set the BaseMacrosOverwrite flag to False for an uninstall when it is necessary to use the current definitions of the macros, rather than their old values.

DisableInstall=*Bool*

If True, this workstation refuses installations.

DisableModifiedTextFilesBAK=*Bool*

If this flag is False, and a file in the [[ModifiedTextFiles]] group is changed, the original file is renamed with a .BK extension. If this flag is set to True, the original file is overwritten by the changed file and no backup is made.

ExcludeFiles=*Bool*

If True, the directory or directories specified in all `ExcludePath` entries in the IPack format file's [[Configuration]] group are not created, nor are any files in those directories copied.

LogAllWarnings=*Bool*

If True, low-priority (nuisance) warnings are logged. By default, only high-priority warnings are logged (and all errors).

OverwriteNewerFiles=*Bool*

Appendix B The P[illegible]ality File

If True, files in the [[AddedFiles]] group are always copied, even if a file with a newer time/date stamp already exists on that workstation. If False, files is copied only if there is no newer existing file.

RemoveIsDelete=*Bool*

If True, the Remove command in the [[ModifiedTextFiles]] group of the IPack format file executes. If set to False, a Remout occurs. For more information, see "[[ModifiedTextFiles]]" in Appendix A.

I claim:

1. A method for installation of an application software package on one or more target workstations from a source workstation, said one or more target workstations and said source workstation all coupled to a network executing a network operating system, said method comprising the steps of:

creating a pre-installation system snapshot of software on said source workstation;

installing said application software package on said source workstation;

creating a post-installation system snapshot of said software on said source workstation;

comparing said pre-installation system snapshot with said post-installation system snapshot;

generating an application installation package based on said step of comparing, said application installation package comprising a plurality of commands for installing said application software package on said one or more target workstations;

storing said application installation package at a storage location accessible by said one or more target workstations and said source workstations;

on each of said one or more target workstations:

saving a predetermined set of system files that will be changed by said application installation package; and

installing said application software package on said corresponding target workstation based on said application installation package.

2. The method as recited in claim 1 wherein said step of comparing said pre-installation system snapshot with said post-installation system snapshot, is further comprised of the steps of:

identifying the differences between said pre-installation system snapshot and said post-installation snapshot; and

generating said plurality of instructions for installing said application software package based on said application software package and the differences between said post-installation snapshot from said pre-installation system snapshot.

3. The method as recited in claim 2 wherein prior to said step of transmitting said application installation file to one or more target workstations performing the step of scheduling said installation of said application software on said one or more target workstations.

4. The method as recited in claim 3 is further comprised of the step of deinstalling said application software from said source workstation.

5. The method as recited in claim 3 wherein said network is a Local Area Network.

6. The method as recited in claim 3 wherein said workstation is operating on a Local Area Network operating system.

7. The method as recited in claim 3 wherein said step of installing said application software package on said corresponding target workstation through said application installation file is further comprised of the step of executing instructions from said application installation file including an instruction for retrieving said application software from said storage location and instructions for modifying system files in said target workstation.

8. A method for installation of an application software package on a target workstation from a source workstation across a network, said method comprising the steps of:

generating a personality file for said target workstation, said personality file describing installation parameters for said application software package and said target workstation;

storing said personality file on said target workstation;

generating an application installation package for said application software package on said source workstation, said step of generating an application installation package further including the steps of:

creating a pre-installation snapshot of said source workstation,

installing said application software package on said source workstation,

creating a post-installation snapshot of said source workstation,

comparing said pre-installation snapshot with said post-installation snapshot; and

generating said application installation package based on differences between said pre-installation snapshot and said post-installation snapshot;

transmitting said application installation package across said network to said target workstation; and

said target workstation receiving said application installation package and installing said application software package based on said application installation package and said personality file.

9. The method as recited in claim 8 wherein said step of said target workstation receiving said application installation package and installing said application software package based on said application installation package and said personality file is further comprised of the step of executing instructions from said application installation package including an instruction for modifying systems files in said target workstation according to said application installation package.

10. The method as recited in claim 9 wherein prior to said step of transmitting said application installation package across said network to said target workstation performing the steps of:

scheduling a predetermined time for the installation of said application software on said target workstation; and

waiting for said predetermined time to be reached.

11. The method as recited in claim 10 is further comprised of the step of deinstalling said application software package from said source workstation.

12. A network for coupling at least a source workstation with a target workstation, said network comprising:

a media for connecting said source workstation with said target workstation;

said source workstation comprising:

a source workstation adapter for coupling said source workstation to said media;

means for creating a pre-installation system snapshot of the software on said source workstation;

means for installing said application software package on said source workstation;

means for creating a post-installation system snapshot of the software on said source workstation;

means for comparing said pre-installation system snapshot with said post-installation system snapshot to identify differences between said pre-installation system snapshot and said post-installation system snapshot;

means for generating an application installation package based on said differences between said pre-installation

system snapshot and said post-installation system snapshot, said application installation package comprising a plurality of commands for installing said application software package on said target workstation;

means for transmitting said application installation package to said target workstation;

said target workstation comprising;

a target workstation adapter for coupling said target workstation to said media;

means for receiving said application installation package; and

means for installing said application software package based on said application installation package.

13. The network as recited in claim **12** wherein said means for comparing said pre-installation snapshot with said post-installation snapshot is further comprised of:

means for determining the differences between said pre-installation snapshot and said post-installation snapshot; and

means for generating said plurality of commands in said application installation package based on differences between said pre-installation snapshot and said post-installation snapshot.

14. The network as recited in claim **13** wherein said source workstation is further comprised of means for assigning a time for transmitting said application installation package to said target workstation.

15. The network as recited in claim **14** wherein said means for receiving said application installation package is a Terminate and Stay Resident program.

16. The network as recited in claim **15** wherein said target workstation is further comprised of means for specifying custom installation parameters.

17. The network as recited in claim **16** wherein said means for installing said application software package based on said application installation package is further comprised of means for installing said application software based on said custom installation parameters.

18. The network as recited in claim **12** further including a means for deinstalling said application software package from said target workstation.

19. A computer-implemented method for generating an application installation package to install an application software package on one or more target workstations from a source workstation, said one or more target workstations and said source workstation all coupled to a network executing a network operating system, said method comprising the steps of:

creating a pre-installation system snapshot of software on said source workstation;

installing said application software package on said source workstation;

creating a post-installation system snapshot of said software on said source workstation;

comparing said pre-installation system snapshot with said post-installation system snapshot;

generating an application installation package based on said step of comparing said pre-installation system snapshot and said post-installation system snapshot, said application installation package comprising a plurality of commands for installing said application software package on said one or more target workstations.

20. The computer-implemented method as recited in claim **19** wherein said step of comparing said pre-installation system snapshot with said post-installation system snapshot further includes the steps of:

identifying differences between said pre-installation system snapshot and said post-installation snapshot; and

generating said plurality of instructions for installing said application software package based on said differences between said post-installation snapshot and said pre-installation system snapshot.

21. The computer-implemented method as recited in claim **20** further including the step of scheduling said installation of said application software on said one or more target workstations.

22. The computer-implemented method as recited in claim **20** wherein said network is an Ethernet Local Area Network.

23. The computer-implemented method as recited in claim **20** wherein said network operating system is NetWare.

24. A computer-implemented method for installation of an application software package on a target workstation from a source workstation across a network, said computer-implemented method comprising the steps of:

generating a personality file for said target workstation, said personality file describing installation parameters for said application software package and said target workstation;

generating an application installation package for said application software package on said source workstation, said step of generating said application installation package further including the steps of:

creating a pre-installation snapshot of said source workstation,

installing said application software package on said source workstation,

creating a post-installation snapshot of said source workstation,

comparing said pre-installation snapshot with said post-installation snapshot, and

generating said application installation package based on differences between said pre-installation snapshot and said post-installation snapshot; and

said target workstation receiving said application installation package and installing said application software package based on said application installation package and said personality file.

25. The computer-implemented method as recited in claim **24** wherein said step of said target workstation receiving said application installation package and installing said application software package based on said application installation package and said personality file further includes the step of executing instructions from said application installation file including an instruction for modifying systems files in said target workstation according to said application installation file.

26. The computer-implemented method as recited in claim **24** further including the step of scheduling a predetermined time for the installation of said application software on said target workstation.

27. The computer-implemented method as recited in claim **24** further including of the step of deinstalling said application software package from said source workstation.

28. A computer system for generating an application installation package to install an application software package on one or more target workstations, said computer system comprising:

means for creating a pre-installation system snapshot of software on said computer system;

means for installing said application software package on said computer system;

means for creating a post-installation system snapshot of said software on said computer system;

means for comparing said pre-installation system snapshot with said post-installation snapshot to identify differences between said pre-installation system snapshot and said post-installation snapshot;

means for generating an application installation package from differences between said pre-installation system snapshot and said post-installation system snapshot, and said application software package, said application installation package comprising a plurality of commands for installing said application software package on said one or more target workstations.

29. The computer system as recited in claim **28** further including means for receiving said application installation package on said one or more target workstations.

30. The computer system as recited in claim **28** further including means for assigning a time for transmitting said application installation package to said one or more target workstations.

31. The computer system as recited in claim **28** wherein said means for receiving said application installation package is a Terminate and Stay Resident program.

32. A computer system for generating an application installation package to install an application software package on one or more target workstations, said computer system comprising:

a program for creating a pre-installation system snapshot of software on said computer system prior to installation of said application software package;

said program also creating a post-installation system snapshot of said software on said computer system after installation of said application software package;

said program comparing said pre-installation system snapshot with said post-installation snapshot to identify differences between said pre-installation system snapshot and said post-installation snapshot; and

said program generating an application installation package from differences between said pre-installation system snapshot and said post-installation system snapshot, and said application software package, said application installation package comprising a plurality of commands for installing said application software package on said one or more target workstations.

33. The computer system as recited in claim **32** wherein said program also assigns a time for transmitting said application installation package to said one or more target workstations.

34. A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

creating a pre-installation system snapshot of software on said computer system prior to installation of an application software package;

creating a post-installation system snapshot of said software on said computer system after installation of said application software package;

comparing said pre-installation system snapshot with said post-installation snapshot to identify differences between said pre-installation system snapshot and said post-installation snapshot; and

generating an application installation package from differences between said pre-installation system snapshot and said post-installation system snapshot, and said application software package, said application installation package comprising a plurality of commands for installing said application software package on one or more target workstations.

35. The machine readable medium as recited in claim **34** wherein said sequences of instructions also cause said computer system to perform the step of assigning a time for transmitting said application installation package to said one or more target workstations.

* * * * *

US006457175B1

(12) **United States Patent**
Lerche

(10) **Patent No.: US 6,457,175 B1**
(45) **Date of Patent: Sep. 24, 2002**

(54) **METHOD AND APPARATUS FOR INSTALLING A SOFTWARE UPGRADE WITHIN A MEMORY RESOURCE ASSOCIATED WITH A COMPUTER SYSTEM**

(75) Inventor: **Robert A. Lerche**, Belmont, CA (US)

(73) Assignee: **Tut Systems, Inc.**, Pleasant Hill, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/189,023**

(22) Filed: **Nov. 9, 1998**

(51) **Int. Cl.**[7] **G06F 9/445**
(52) **U.S. Cl.** **717/173**; 717/170; 717/178; 709/221; 710/64; 710/74
(58) **Field of Search** 713/2, 100; 709/221, 709/222; 710/74, 72, 55, 64, 68, 10; 380/251; 463/42, 43; 717/168, 170, 173, 174, 178

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,699,275	A *	12/1997	Beasley et al.	709/221
5,701,492	A *	12/1997	Wadsworth et al.	717/11
5,759,102	A *	6/1998	Pease et al.	463/42
5,809,251	A *	9/1998	May et al.	709/223
5,848,064	A *	12/1998	Cowan	370/338
5,852,735	A *	12/1998	Urban	717/11
5,870,609	A *	2/1999	Thornton et al.	717/11
5,896,566	A *	4/1999	Averbuch et al.	455/419
5,905,523	A *	5/1999	Woodfield et al.	348/12
5,999,741	A *	12/1999	May et al.	717/11
6,047,128	A *	4/2000	Zander	717/11
6,151,657	A *	11/2000	Sun et al.	711/103
6,301,480	B1 *	10/2001	Kennedy, III et al.	455/445
6,311,291	B1 *	10/2001	Barrett, Sr.	714/25

OTHER PUBLICATIONS

"Concurrent Operations on the Database", *Principles of Database Systems*, Computer Science Press, 1982, pp 394–399.

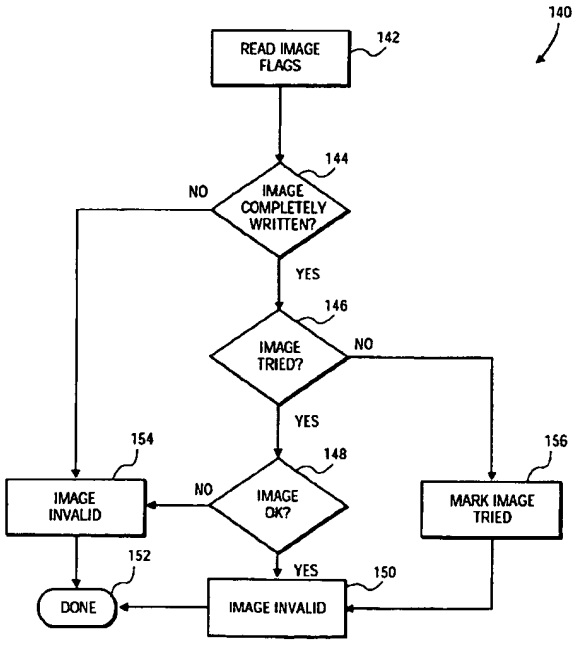
* cited by examiner

Primary Examiner—Tuan Q. Dam
(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

A method of installing a software application image, for a software upgrade, within a remote and embedded target device includes the step of storing both a current and an upgraded software application image within an EEPROM within the target device. The set of instructions embodied within the current application image is maintained within the target device, while a validation operation is performed with respect to a set of instructions embodied within the upgraded software application image. Only once complete installation and successful execution of the upgraded application image have been validated is the upgraded application image designated as a current application image, and the previously installed application image discarded. Accordingly, the risk of the target device being a rendered unbootable as a result of the installation of a software upgrade is reduced.

32 Claims, 9 Drawing Sheets



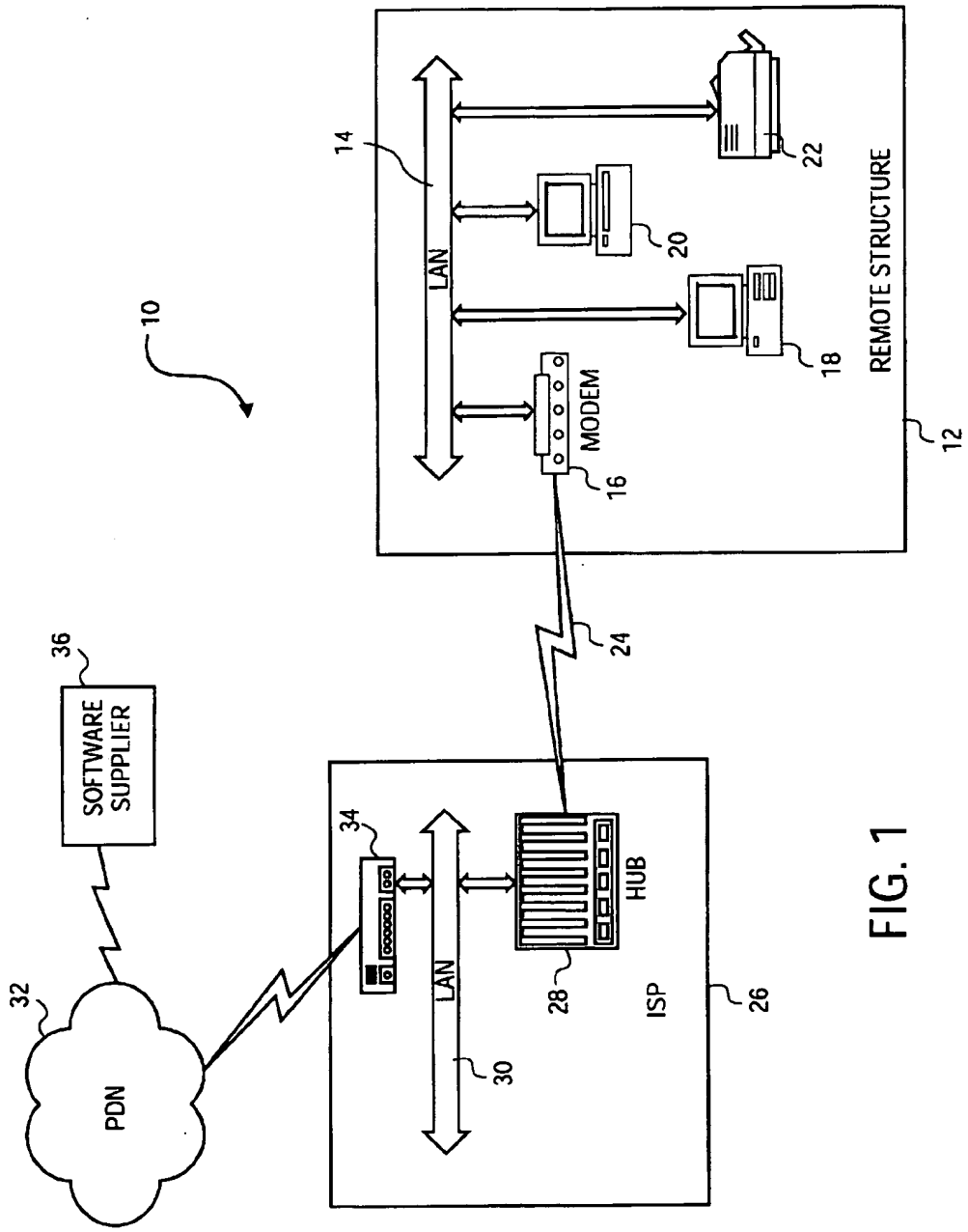

10
14
LAN
22
20
18
MODEM
16
REMOTE STRUCTURE
12
24
36
SOFTWARE SUPPLIER
34
HUB
LAN
28
ISP
26
30
32
PDN

FIG. 1

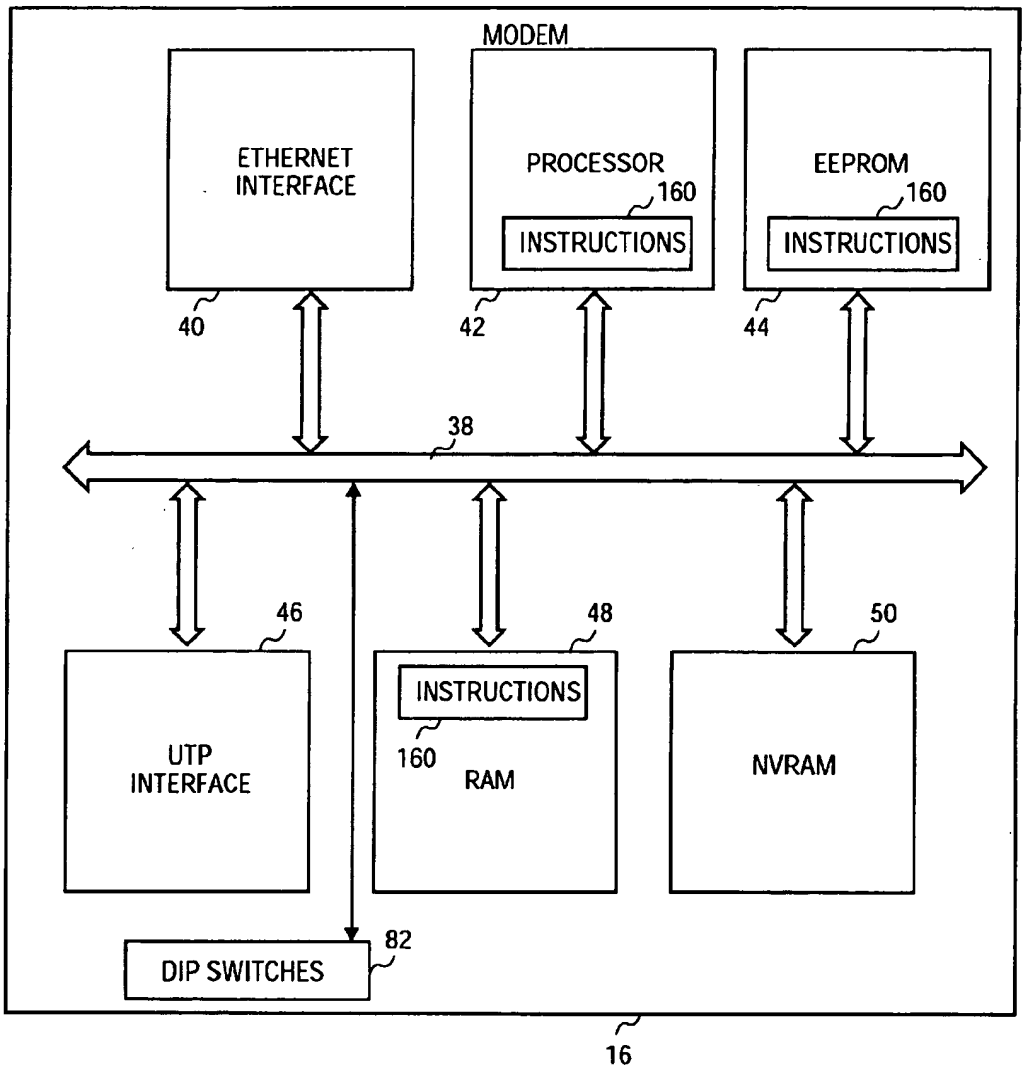

MODEM
ETHERNET INTERFACE
PROCESSOR
160
INSTRUCTIONS
EEPROM
160
INSTRUCTIONS
40
42
44
38
46
48
50
INSTRUCTIONS
160
RAM
UTP INTERFACE
NVRAM
DIP SWITCHES
82
16

FIG. 2

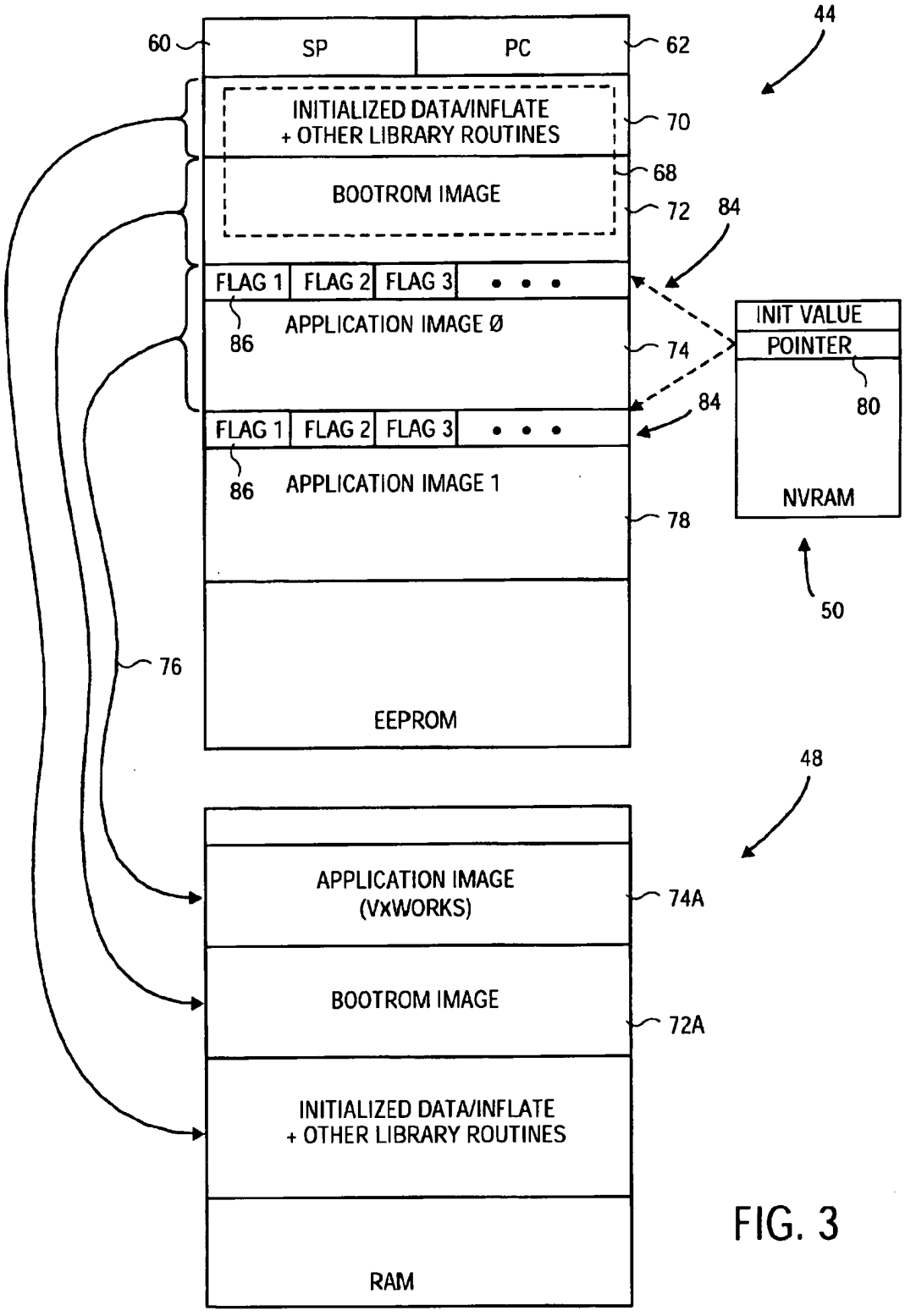

44
60
SP
PC
62
INITIALIZED DATA/INFLATE
+ OTHER LIBRARY ROUTINES
70
BOOTROM IMAGE
68
72
84
FLAG 1
FLAG 2
FLAG 3
• • •
86
APPLICATION IMAGE Ø
74
INIT VALUE
POINTER
80
NVRAM
84
FLAG 1
FLAG 2
FLAG 3
• • •
86
APPLICATION IMAGE 1
78
50
76
EEPROM
48
APPLICATION IMAGE
(VxWORKS)
74A
BOOTROM IMAGE
72A
INITIALIZED DATA/INFLATE
+ OTHER LIBRARY ROUTINES
RAM

FIG. 3

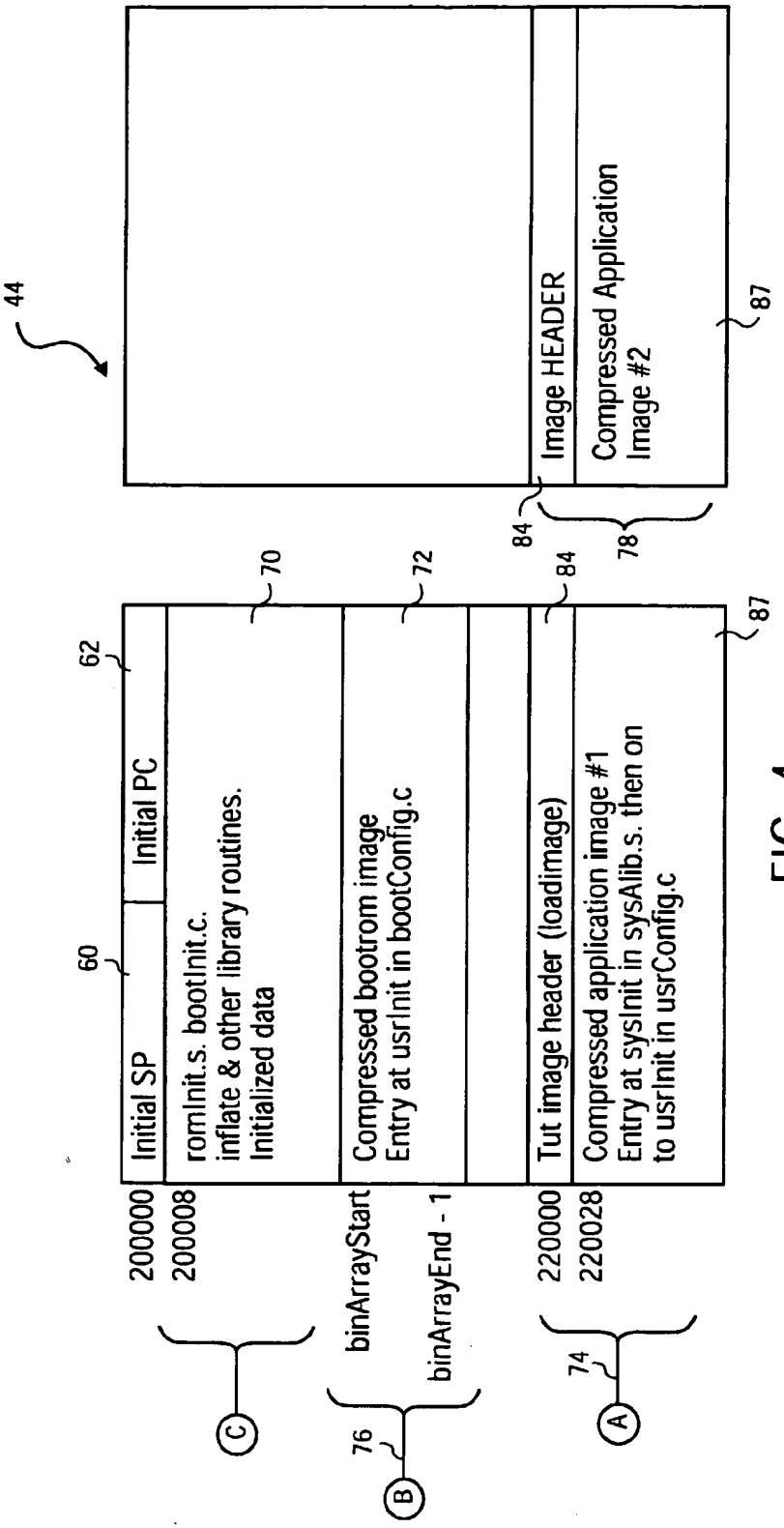

44
Image HEADER
Compressed Application
Image #2
87
84
78
70
72
84
87
62
60
Initial SP
Initial PC
romInit.s. bootInit.c.
inflate & other library routines.
Initialized data
Compressed bootrom image
Entry at usrInit in bootConfig.c
Tut image header (loadimage)
Compressed application image #1
Entry at sysInit in sysAlib.s. then on
to usrInit in usrConfig.c
200000
200008
binArrayStart
binArrayEnd - 1
220000
220028
74
76
C
B
A

FIG. 4

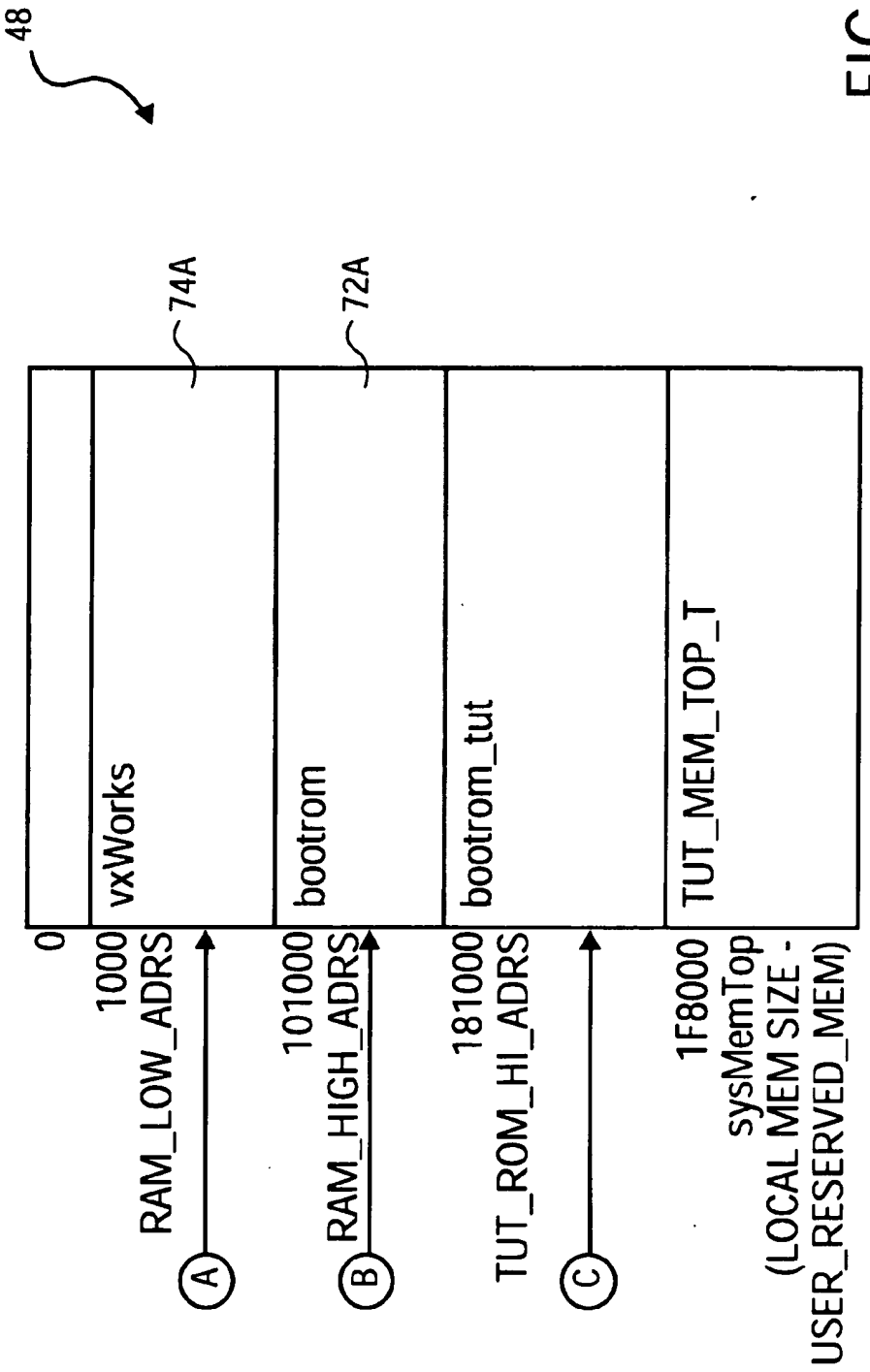

48
74A
72A
vxWorks
bootrom
bootrom_tut
TUT_MEM_TOP_T
0
1000
RAM_LOW_ADRS
101000
RAM_HIGH_ADRS
181000
TUT_ROM_HI_ADRS
1F8000
sysMemTop
(LOCAL MEM SIZE -
USER_RESERVED_MEM)
A
B
C

FIG. 5

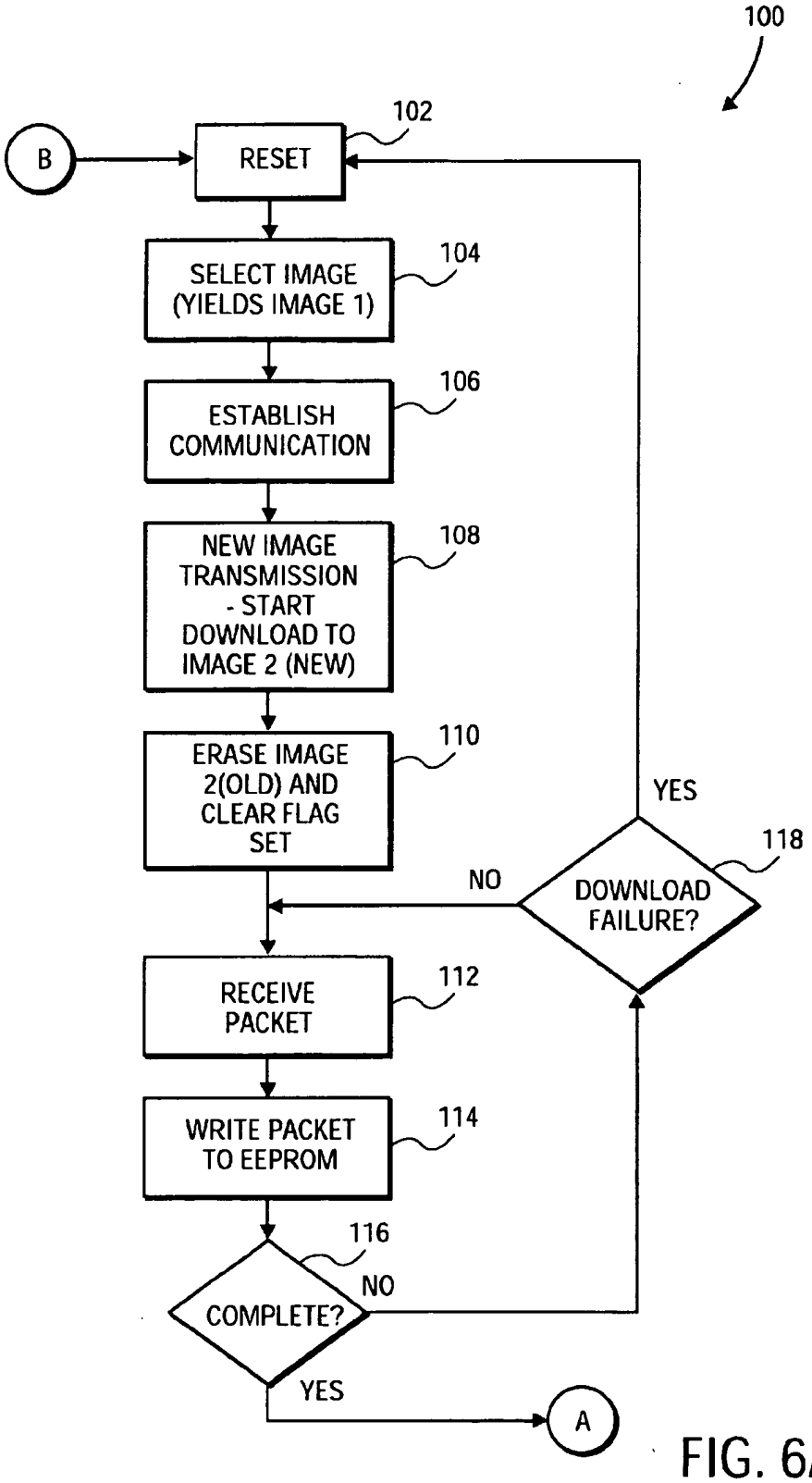

100
B
RESET
102
SELECT IMAGE (YIELDS IMAGE 1)
104
ESTABLISH COMMUNICATION
106
NEW IMAGE TRANSMISSION - START DOWNLOAD TO IMAGE 2 (NEW)
108
ERASE IMAGE 2(OLD) AND CLEAR FLAG SET
110
DOWNLOAD FAILURE?
118
YES
NO
RECEIVE PACKET
112
WRITE PACKET TO EEPROM
114
COMPLETE?
116
NO
YES
A

FIG. 6A

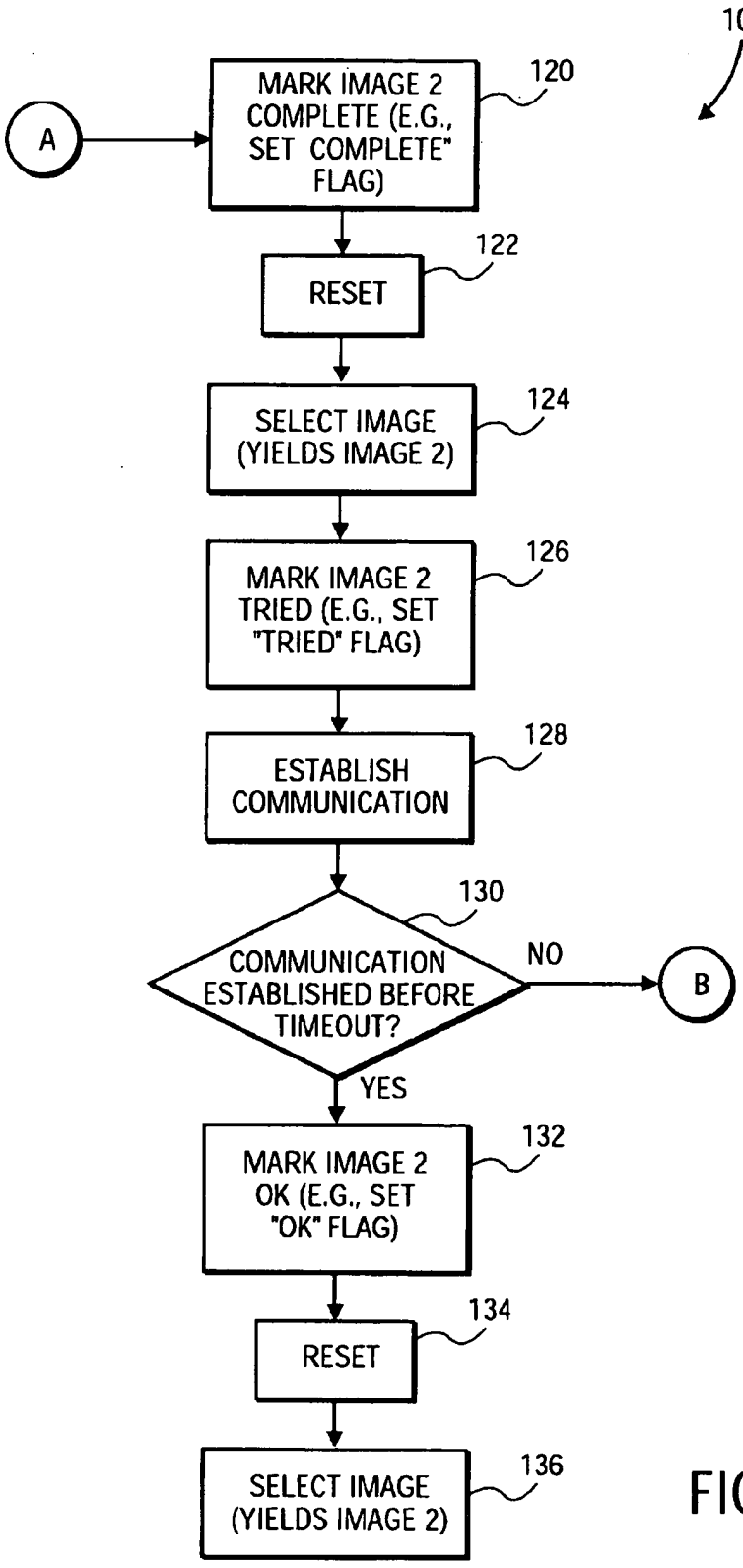

100
A
MARK IMAGE 2 COMPLETE (E.G., SET COMPLETE" FLAG)
120
RESET
122
SELECT IMAGE (YIELDS IMAGE 2)
124
MARK IMAGE 2 TRIED (E.G., SET "TRIED" FLAG)
126
ESTABLISH COMMUNICATION
128
COMMUNICATION ESTABLISHED BEFORE TIMEOUT?
130
NO
B
YES
MARK IMAGE 2 OK (E.G., SET "OK" FLAG)
132
RESET
134
SELECT IMAGE (YIELDS IMAGE 2)
136

FIG. 6B

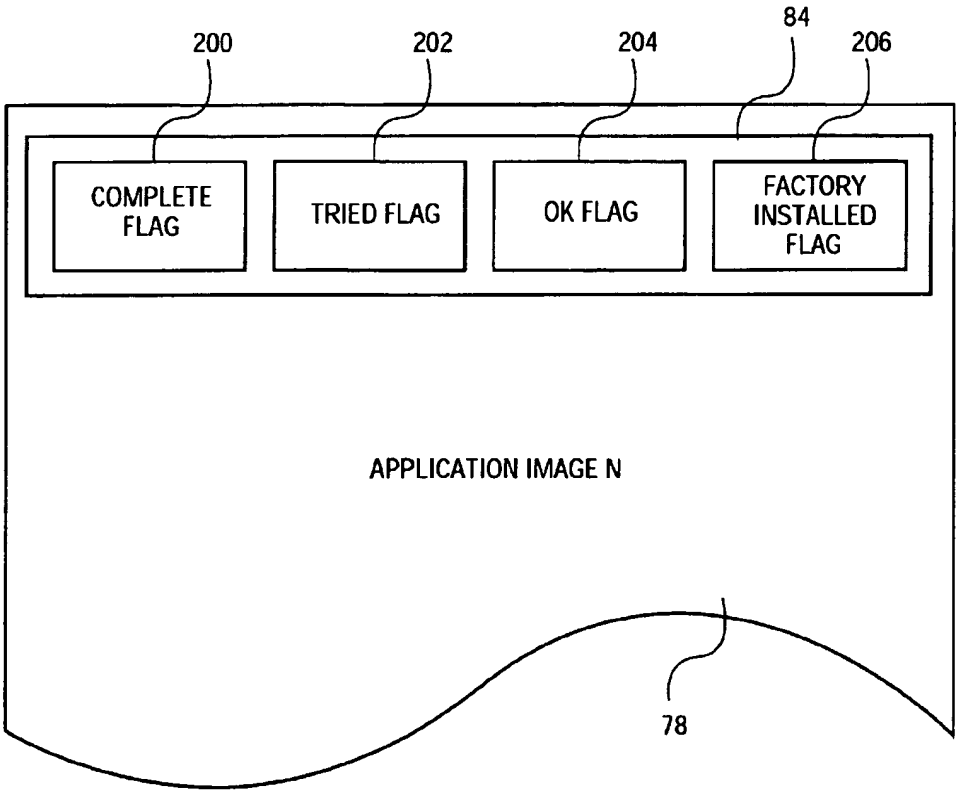

200
202
204
84
206
COMPLETE FLAG
TRIED FLAG
OK FLAG
FACTORY INSTALLED FLAG
APPLICATION IMAGE N
78

FIG. 7

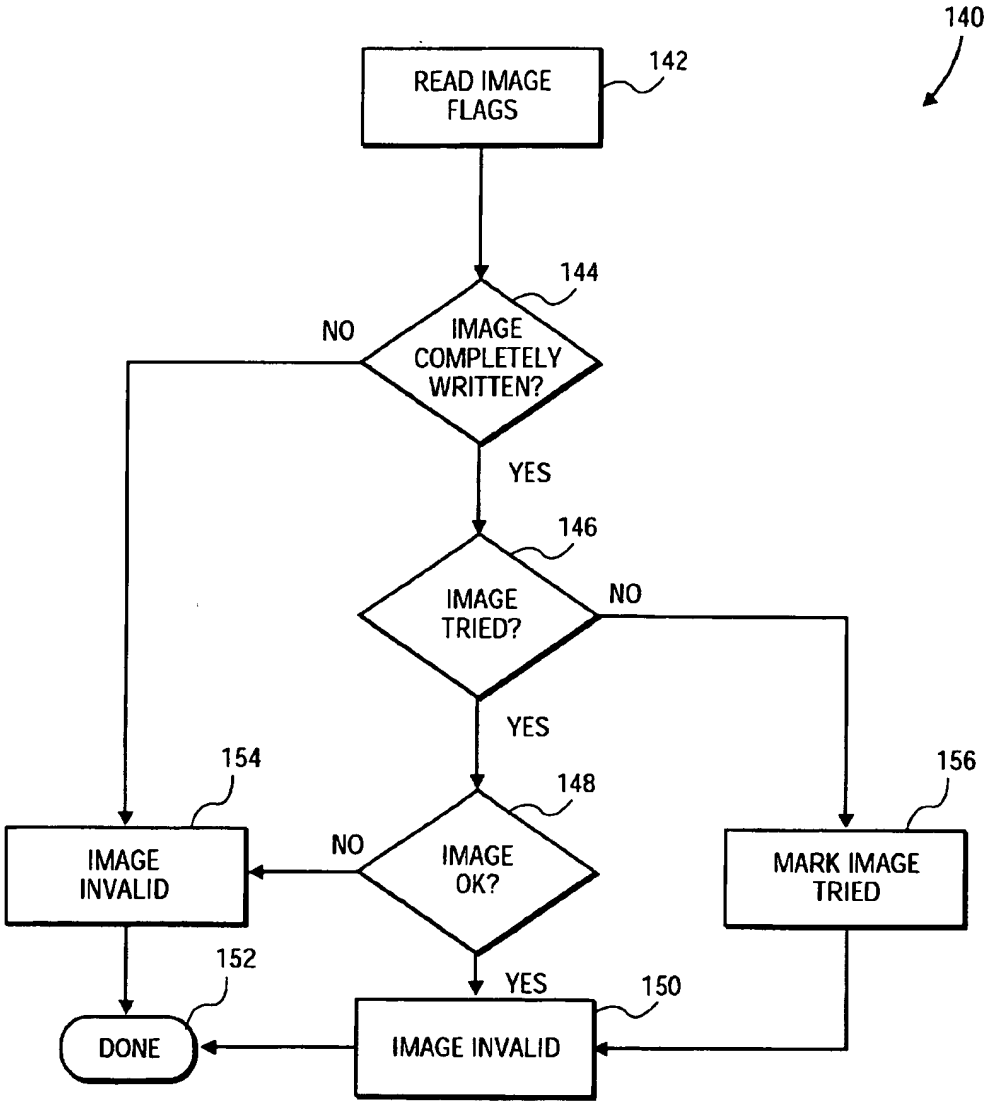

140
READ IMAGE FLAGS
142
144
IMAGE COMPLETELY WRITTEN?
NO
YES
146
IMAGE TRIED?
NO
YES
148
IMAGE OK?
NO
YES
154
IMAGE INVALID
156
MARK IMAGE TRIED
152
DONE
150
IMAGE INVALID

FIG. 8

METHOD AND APPARATUS FOR INSTALLING A SOFTWARE UPGRADE WITHIN A MEMORY RESOURCE ASSOCIATED WITH A COMPUTER SYSTEM

FIELD OF THE INVENTION

The present invention relates generally to the field of software installation and, more specifically, to the field of replacing a first set of instructions stored within a memory resource associated with a machine, with a second set of instructions for execution by the machine.

BACKGROUND OF THE INVENTION

Microprocessors and microcontrollers are being used increasingly in embedded applications to provide operational and functional intelligence within a wide variety of devices. It will be appreciated that the software stored within memories associated with such microprocessors and microcontrollers may require upgrading from time to time to provide increased or modified functionality to a device in which the microprocessor or microcontroller is embedded. With the proliferation of networks in a wide variety of applications, such as the home environment, software installed on a networked device may often conveniently be upgraded from a remote location via the network. However, it may occur that a networked device is positioned in a location which is difficult to access, or even inaccessible. The remote upgrading of software stored within a memory associated with the networked device may be problematic in that, should the upgrade installation fail for some reason, the networked device may be rendered totally inoperative. The restoration of functionality to the network device may in such cases be expensive and inconvenient. This is especially true when the networked device is being upgraded from a remote location, and service personnel are required to be dispatched to the site of the networked device to address and correct the failed software upgrade operation.

SUMMARY OF THE INVENTION

According to the invention, there is provided a method of installing a second set of instructions within a machine, the machine including a memory resource storing a first and the second set of instructions. The machine executes the first set of instructions, and validates the second set of instructions. If the second set of instructions is validated, the second set of instructions is then indicated as being executable in place of the first set of instructions.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not limitation, in the figures of the accompanying drawings in which like references indicate similar elements.

FIG. **1** is a diagrammatic representation of a networked environment within which the present invention may be employed.

FIG. **2** is a block diagram illustrating an exemplary machine in the form of a modem within which the present invention may be employed.

FIG. **3** is a block diagram illustrating an exemplary system memory layout, according to one embodiment of the present invention, that may be implemented within the memory of the machine illustrated in FIG. **2**.

FIGS. **4** and **5** are block diagrams providing further details regarding a memory layout for an exemplary embodiment of the present invention.

FIGS. **6**A and **6**B are flowcharts illustrating a method, according to one exemplary embodiment of the present invention, of installing a set of instructions, in the form of an application image, within a machine.

FIG. **7** is a diagrammatic representation of an exemplary application image, illustrating a flag set associated with the application image.

FIG. **8** is a flow chart illustrating a method, according to one exemplary embodiment of the present invention, of determining whether an object code image stored within a memory resource is valid.

DETAILED DESCRIPTION

A method and apparatus for installing a software upgrade within a memory resource associated with a computer system are described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident, however, to one skilled in the art that the present invention may be practiced without these specific details.

FIG. **1** is a diagrammatic representation of an exemplary network environment **10** within which the present invention may be employed. Specifically, a remote structure **12**, for example a small office or home, is shown to accommodate a local area network (LAN) **14** that networks a modem **16**, a personal computer **18**, a Macintosh **20**, and a printer **22**. The LAN **14** may be implemented utilizing Unshielded Twisted Pair (UTP) wiring, power supply wiring, dedicated network wiring, or infrared connections, within the remote structure **12**. In one embodiment, the LAN **14** is implemented over UTP wiring within the remote location **12** using the HomeRun™ networking technology developed by Tut Systems, Inc. of Pleasant Hill, Calif. The modem **16** is shown to establish a communication link **24** to an Internet Service Provider (ISP) **26** and, more specifically, to a hub **28** operated by the ISP **26**. The communications link **24** may be implemented over conventional plain old telephone service (POTS) wiring, and may comprise a Digital Subscriber Line (DSL) link. Alternatively, the communications link may be established via radio frequency communications, or satellite. The hub **28** may include a number of cards (not shown) such as, for example, a management card, a line card, and a switch card. Each of these cards may carry a processor, such as the 68360 microprocessor manufactured by Motorola Inc.. The ISP **26** also implements a LAN **30**, via which the hub **28** is networked or coupled to the public data network (PDN) via a router **34**.

FIG. **2** is a block diagram illustrating an exemplary embodiment of the composition of the modem **16**. Specifically, the modem **16** is shown to include a bus **38** via which the various components of the modem **16** communicate and are interfaced. The modem **16** also includes an Ethernet interface **40** via which the modem **16** communicates over the LAN **14**. In one embodiment, the Ethernet interface **40** facilitates 10baseT communications over the LAN **14**. A processor **42**, for example a 68302 processor manufactured by Motorola Inc., an electrically erasable programmable read-only memory (EEPROM) (or flash memory) **44**, a UTP interface **46**, a random access memory (RAM) **48** and a non-volatile random access memory (NVRAM) **50** are also all shown to be coupled to the bus **38**. The EEPROM **44** and the RAM **48** store sequences of

instructions that are executed by the processor **42** and that control operation of the modem **16**.

It will be appreciated that, from time to time, the manufacturer of the modem **16** may wish to release upgraded software for installation and execution within the modem **16**. Typically, such an upgrade operation requires that the upgraded software be loaded into and stored within at least the EEPROM **44**. One option would be to distribute the software to the user of the modem **16** (e.g., on a diskette or CD-ROM), and have the user manually install the upgraded software. As will be appreciated from FIG. **1**, a software supplier **36** coupled to the PDN **32** may remotely distribute and install the upgraded software on the modem **16**, as this may provide a number of operational efficiencies. While the downloading and installation of software within the modem **16** via the communications link **24** may be performed without end-user intervention, there is a risk that the installation of the upgraded software may be unsuccessful. In a worst-case scenario, the attempted upgrade may in fact place the modem **16** in a condition in which it cannot boot at all. Specifically, wherein the downloaded software is not executable, for example as a result of a corrupt transmission, and the corrupt software overwrites previously installed software, the modem **16** may not be revivable from a remote location. In such as situation, manual intervention by the software supplier **36** or ISP **26** may be required to return the modem **16** to an operational state. Where the software supplier has a large installed base of modems, such manual intervention may be prohibitively expensive.

With a view to avoiding situations within which manual intervention may be required to restore functionality, and specifically the ability to boot, to a target machine such as a computer system, the present invention proposes a software download mechanism that prevents a target of the download operation from being placed in the state in which it cannot boot at all. In summary, the present invention teaches maintaining at least two software images within a memory resource associated with a processor of a computer system. Specifically, a first of the software images is an image of software that has been validated and is known to be executable on the target machine. A second software image is only validated for execution after it is known to run successfully, and to perform at least one predetermined task (e.g., to establish communication with a network device following a reset). The first ("old") image is retained within the memory resource in case the second ("new") image fails to execute. The present invention is advantageous in that a software image that is known to be executable is always maintained within the memory resource, so that the target device is never placed in a situation in which it cannot boot. Specifically, in the case that the second "new" software image is unexecutable, for whatever reason, the target machine may always be booted from the first "old" software image.

For the purposes of the present specification, the term "software image" shall be taken to mean the output of a linker (also termed a link editor or binder) that combines object modules to form an executable program. The term "set of instructions" shall be taken to include the term "software image".

While the below description describes an exemplary embodiment of the invention in the context of the networked environment **10** shown in FIG. **1**, and specifically as implemented within the modem **16**, it will be appreciated that the teachings of the present invention are applicable to a wide range of environments and applications. The teachings of the present invention are applicable to any situation or environment in which an upgraded or second version of a software program, or portion of such a program, is installed within or on a machine. For example, the upgrading of software installed on remote Internet access devices, such as x2 modems, or web access devices such as WebTV, may utilize the teachings of the present invention. Further, the teachings of the present invention may be applied to facilitate reliable upgrading of software within machines executing Java code.

Referring now to FIG. **3**, there is illustrated an exemplary memory layout that may be implemented within the EEPROM **44** and RAM **48** of the modem **16**. Referring first to the EEPROM **44**, a stack pointer (SP) **60** and a program counter (PC) **62** are stored at the lowest address within the EEPROM **44**. The stack pointer **60** and the program counter **62** are retrieved by hardware upon reset or power up of the modem **16**, and identify the location at which the execution of programs stored within the EEPROM **44** is to begin. Adjacent the memory location at which the stack pointer and the program counter **60** and **62** are stored, boot code **68** is stored within the EEPROM **44**. The boot code **68** comprises initialized data and inflate and other library routines **70**, as well as a boot EPROM image **72**. The boot code **68** initializes the general registers of the processor **42**, and initializes other hardware modules that allow the modem **16** to setup memory and to communicate with external devices. Immediately below the boot code **68**, a first application image **74** is stored within the EEPROM **44**. In one embodiment, the application image **74** comprises a combination of (1) an embedded, real-time operating system, such as the VxWorks operating system developed by Wind River Systems of Alameda, Calif. and (2) application software for operation of the modem **16**. While the application image **74** may in certain embodiments be executable from within the EEPROM **44**, access to the RAM **48** is faster, and it may thus be desirable to move the application image **74** from the EEPROM **44** to the RAM **48**, as indicated by arrow **76**. The application image **74** may furthermore be compressed, in which case the application image **74** is decompressed before being stored within the RAM **48**. The boot code **68**, and specifically the inflate routine, is responsible for performing the above transfer and decompression operations.

A second application image **78** is stored within the EEPROM **44** immediately below the first application image **74**. The second application image may comprise an updated or modified version of the operating system and application software embodied in the first application image **74**. Alternatively, the first application image **74** may comprise an upgraded version of the operating system and application software embodied in the second application image **78**. In any event, either one of the application images **74** or **78** may be designated as a "current" application image, which is decompressed and transferred to the RAM **48** for execution. In one embodiment, the current application image is identified by a pointer value **80** stored within the NVRAM **50**. The non-current application image may be a new application image that has been downloaded and stored in the EEPROM **44**, but not as yet validated, or an old application image for which the current application image is a replacement. For example, the pointer **80** may point to a current and executable "old" application image, while upgraded "new" software, in the form of a second application image, may be written to a second application image location within the EEPROM **44**. Upon successful validation of the "new" application image, the pointer **80** may then be switched to indicate the "new" application image as the current application image for execution in place of the "old" application image. Any further software upgrade downloads may then

be written over the "old" application image, as the executability of the new software image will have been verified, and the danger of rendering the machine unbootable is reduced.

As illustrated in FIG. **3**, each of the application images **74** and **78** includes a header portion **84** including a number of the flags **86**. In one embodiment, each of the flags **86** may represent and indicate the completion of a validation step with respect to the associated application image for example, a sequence of flags may indicate that the application image **74** or **78** has the been completely written to the EEPROM **44**, that the application image has been utilized for an application boot operation at least once, that is the modem **16** has performed at least one predetermined operation under the direction of the application image, or that the application image is factory installed.

With respect to the predetermined operation mentioned above, the modem **16** may, merely for example, under the direction of an as yet unvalidated software image, attempt to perform a sequence of steps required for the establishment of a valid connection from the modem **16** to the ISP **26**. These steps may involve booting the modem **16**, turning on a DSL line, performing a set of the synchronization steps with respect to the ISP **26**, sending a predetermined message to the ISP **26**, and receiving a response from the ISP **26**. Upon receipt of the message from the ISP **26**, a flag indicating successful completion of the predetermined operation may be toggled.

In one embodiment of the present invention, the application images may conveniently be written into the EEPROM **44** so that sector boundaries divide the application images **74** and **78**. This allows an application image and associated flags to be erased in a single step by writing all bits within a relevant sector to one (1). In this case, the flags may be set by setting a flag from 1 to 0 (that is, the flags are set low).

A more detailed description of an exemplary embodiment of the present invention, and specifically of a system memory layout and boot-up procedure of a target device, such as a the modem **16**, will now be described with reference to FIGS. **4** and **5**. The exemplary embodiment will be described in the context of the VxWorks operating system developed by Wind River Systems, Inc. However, the broad teachings of the present invention could also be implemented utilizing real-time operating systems (RTOS) such as the QNX operating system developed by QNX Software System Limited, or the Windows C.E. operating system developed by Microsoft Corp. of Redmond, Wash.

The VxWorks operating system supports a wide variety of target memory configurations. One exemplary memory configuration is described below. It will readily be appreciated that the principles described below may equally well be implemented in an alternative memory configuration. Referring now specifically to FIG. **4**, the EEPROM **44** is shown to include an initial stack pointer (SP) **60** and an initial program counter (PC) **62**. The EEPROM **44** stores a VxWorks development boot EEPROM image **72** below the stack pointer **60** and the program counter **62**, the boot ROM image **72** functioning to load a test version of a target application via the Ethernet interface **40**. The boot ROM image **72** is a special VxWorks application that reads in a link module and branches to it, restarting the target (i.e., the modem **16**). Besides loading a test version of an application, the boot ROM image **72** can also be executed to set or display memory, set boot parameters, set an Ethernet MAC address in the NVRAM **50**, or launch to an arbitrary memory address. Depending on boot parameters, it either loads an application image for execution after a delay, or runs a simple command loop. In the modem **16**, even though the development boot ROM image **72** is present, it will not necessarily be executed during normal operation as a romStart routine, to be described below, selects an application image instead.

Below the development boot ROM image **72**, a compressed application ROM image **74** is stored, the application ROM image **74** containing an application image **74A** to be copied directly to the RAM **48**, as indicated by the arrow **76** at power-on reset time. As RAM **48** access is typically faster than EEPROM **44** access, application images typically are not executed directly out of the EEPROM **44**. However, the VxWorks operating system supports a ROM resident configuration, which may be utilized in the present invention where speed is less important than the conservation of RAM capacity. The application image **74A** is stored at a RAM_LOW_ADDRESS location within the RAM **48**, and both ethernet-loaded and ROM-loaded applications begin execution at this address. It will be noted that the boot ROM image **72A**, transferred from the compressed boot ROM image **72**, is stored at RAM_HIGH_ADDRESS, thus providing room within the RAM **48** for the application image **74A** to be accommodated between the RAM_LOW_ADDRESS and the RAM_HIGH_ADDRESS. In one exemplary embodiment, the modem **16** may include a second ROM that holds a second application ROM image **78**, a pointer **80** in the NVRAM **50** indicating either the image **74** or **78** as being a current image for execution by the processor **42**. In an alternative embodiment of the present invention, as illustrated in FIG. **3**, a single ROM may accommodate both images **74** and **78**. In either case, it should be noted that two application images **74** and **78** are stored and are transferable to the RAM **48** for execution purposes.

The boot-up, or starting, of the modem **16** will now be described. The program counter **62** indicates an initial entry point following a modem **16** power-on reset. Specifically, the entry point is indicated as being the romInit routine stored in romInit.s. The romInit routine does basic hardware initialization, set up the processor stack, and branches to a romStart routine in the bootInit.c. The romStart routine is the first C code to execute, and functions to clear memory, copy a current application image from the EEPROM **44** to the RAM **48**, and to jump to a RAM **48** entry point. In one embodiment, a standard VxWorks configuration positions a romStart routine image at either the RAM_LOW_ADDRESS or the RAM_HIGH_ADDRESS, opposite the application image **74A**. Although linked to such a RAM address, the romStart routine begins execution within the EEPROM **44**. Specifically, the romStart routine first calls a copyLong routine, and copies the romStart routine image text and data to the RAM address. After copying the start-up code and clearing the rest of the RAM **48** to zero's, the romStart routine then decompresses the application image **74A**, and branches to it. The romStart routine is furthermore responsible for selecting which of the application images **74** or **78** to execute based on the pointer **80** stored within the NVRAM **50**, the state of the images indicated by the image headers **84**, and possibly an external input such as a setting of the dip switches **82**.

As illustrated in FIG. **4**, each of the application images **74** and **78** includes both an image header portion **84** and a compressed body portion **87**. Each of the application image header portion **84** includes a number of flags **86** providing information concerning the associated application image. Specifically, the flags **86** may indicate whether the application image is complete (that is, whether the application

image was successfully written into the EEPROM **44**); whether execution of the application image has been attempted once; and whether the modem **16** runs successfully under the application image (e.g., communicates with a line card within the hub **28** of the ISP **26**). The image header portion **84** may also include flags **86** indicating the size of the image, the type of the image, and a string giving the software revision name. In one embodiment, an initial application image that is factory installed may include an additional flag marking the image as factory installed, and indicating that the image is expected to work, and should be tried first. Accordingly, the romStart routine may examine the image header portions **84** associated with both images **74** and **78** to determine which of these application images is current. For example, the flag **86** indicating that the application image runs successfully will not be set for a newly loaded application image which has not been validated.

FIGS. **6**A and **6**B are flowcharts illustrating a method **100**, according to one exemplary embodiment of the present invention, of installing a second set of instructions, in the form of an application image, within a machine, such as for example the modem **16**. The method **100** commences at step **102** with a reset operation, and proceeds to step **104**, where the romStart routine selects a first application image, for example application image **74**, as a current image to boot. This determination is made by the romStart routine with input from the NVRAM **50** in the form of the pointer **80** which identifies the application image **74** as the current image.

At step **106**, the modem **16** performs a validation operation. While this validation operation may vary from embodiment to embodiment of the present invention, in the present embodiment, the validation operation comprises the step of establishing a communication link between the modem **16** and the ISP **26**. Specifically, the establishment of the communication link at step **106** may include booting the modem **16** utilizing the first application image **74** selected at step **104**, turning on a Digital Subscriber Line (DSL), synchronizing the modem **16** and ISP hardware, sending a message from the modem **16** to the ISP **26**, and receiving a response message from the ISP **26** at the modem **16**. It will however be appreciated that, in other applications, any number of other validation operations may be performed to confirm proper operation of the device when executing a set of instructions comprising any particular application image. Having established a communication link at step **106**, the transmission of a new, second application image **78** to the modem **16** is commenced at step **108**. The File Transfer Protocol (FTP) may be utilized for the transmission of the second application image **78** to the modem **16**. For example, the second application image **78** may be propagated to the modem **16** from the software supplier **36** over the PDN **32** and via the ISP **26**. In the event that the second application image **78** is to be written to a memory location occupied by a third application image (e.g., an application image that was replaced by the first application image selected at step **104**), the third application image is erased, and associated flags **86** are cleared, at step **110**. At step **112**, a packet included within the transmission begun at step **108** of the new, second application image **78** is received at the modem, and this packet is then written to the EEPROM **44** at step **114**. The second application image **78** may be received, merely for example, within the modem **16** via the Ethernet interface **14** from the LAN **14**. Alternatively, the second application image **78** may be received into the EEPROM **44** via the UTP interface **46** via the communication link **24** from an external software supplier **36**. Referring to FIG. **3**, the second application image **78** is written to a separate, distinct memory within the EEPROM **44**, so as not to override the first application image **74**. At decision box **116**, a determination is made as to whether the download of the second application image **78** is complete. This determination is made, for example, by referencing indications provided by the File Transfer Protocol (FTP). If not, the method **100** to proceeds to decision box **118**, wherein a determination is made as to whether a download failure has occurred. For example, a packet comprising the transmission to the modem **16** may have been corrupted, or not received. If a download failure has occurred, the method **100** then loops back to step **102**, where the reset operation is again performed, and the modem **16** boots using the first image **74**. Alternatively, should no download failure **118** be detected at decision box **118**, the method **100** loops back to step **112**, where a further packet of the transmission of the new application image **78** is received. Assuming no download failure, the method **100** loops through steps **112**–**118** until the complete second application image **78** has been written into the EEPROM **44**.

Once it is determined that decision box **116** that the transmission and reception of the second application image **78** has been completed, the method **100** proceeds to step **120**, as indicated in FIG. **6**B, where the second application image **78** is marked as complete. Referring now to FIG. **7**, an exemplary second application image **78** is illustrated. The application image **78** includes a header portion **84** including a number of flags **86**, namely:

1. A "complete" flag **200**;
2. A "tried" flag **202**;
3. An "okay" flag **204**; and
4. A "factory installed" flag **206**.

The second application image **78** may be marked as complete at step **120** by setting the complete flag **200** to a predetermined state. The setting of the complete flag **200** indicates that the associated second application image **78** is complete and has been completely and successfully written into the EEPROM **44**. At step **122**, the modem **16** is then again subject to a reset operation. At step **124**, the second application image **78** is selected for the boot-up operation. This selection may be performed by temporary indicating the second application image **78** as the current application image within the NVRAM **50**. At step **126**, the second application image **78** is marked as "tried". For example, the step **126** may include setting the tried flag **202** to a predetermined state. At step **128**, the modem **16**, under the direction of the set of instructions comprising the second application image **78**, attempts to perform a validation operation to validate operation of the modem **16**. In one exemplary embodiment, the validation operation may again comprise the establishment of a communication link between the modem **16** and the ISP **26** in the manner described above with reference to step **106**. At decision box **130**, a determination is made as to whether the communication link has been established before a timeout period. For example, should a "timeout" timer (not shown) within the modem **16** expire, the condition proposed at decision box **130** will not be met, in which case the method **100** returns to step **102**, where the modem **16** is reset.

On the other hand, should a communication link the successfully established between the modem **16** and the ISP before the timeout, the second application image **78** is marked as valid at step **132**, for example, by setting the OK flag **204** to a predetermined state. Accordingly, the flags **200**, **202**, and **204** will each have been sent to a predetermined state to indicate the second application image **78** as valid.

The setting of all three flags **200**, **202** and **204** to the predetermined state marks the associated application image as a validated boot-up image, and accordingly the pointer **80** within the NVRAM **50** will retain an indication of the application image selected at step **124** as the boot-up image. At step **134** the modem **16** is again subject to a reset operation, whereafter the second application image **78** is selected as the boot-up image at step **136** in accordance with the indication provided by the pointer **80**, and on account of the flags **200**, **202**, and **204** for the second application image **78** indicating the second application image **78** as being successfully validated.

Turning now to FIG. **8**, there is illustrated a flowchart illustrating a method **140**, according to an exemplary embodiment of the present invention, of determining whether an application image is valid by the inspection of various flags **86** included within a header portion **84** associated with the relevant application image. The method **140** may be executed at any one of the steps of method **100** that require the selection of an application image. Specifically, the step of selecting an application image for a boot-up operation may be performed by examining an application image indicated by the pointer **80**, and then by determining whether the application image is valid utilizing the method **140**.

The method **140** commences at step **142** where the various flags **86** included within the header portion **84** are read. At decision box **144**, a determination is made as to whether the complete flag **200** (as illustrated in FIG. **7**) indicates that the application image is complete. Specifically, a determination may be made as to whether the complete flag **200** has been set to a logical zero (0). If so, a further determination is then made at decision box **146** whether the image has previously been tried. Specifically, the tried flag **202** is examined to determine whether it has been set, for example, to a logical zero (0). If the image has not been tried (i. e., is newly installed but not yet validated), step **156** marks the image "tried" and proceeds to step **150** indicating that the image is to be executed. If the image has been tried as indicated by flag **202** execution proceeds to decision box **148**. Decision box **148** makes a final determination based on the OK flag **204**. Specifically the OK flag **204** may be examined to determine whether it has been set, for example, to a logical zero (0). If the OK flag **204** is set, the image has previously successfully performed a predetermined validation operation and hence is valid for execution. The method **140** accordingly recognizes the image as valid at step **150** and terminates at step **152**. If the OK flag **204** is not set the method **140** determines that the validation has been attempted but failed, recognizes the image as invalid at step **154**, and terminates at step **152**.

The method **140** proposed by the present invention is advantageous in that it does not permit a target device to view an application image as being valid, and therefore selectable as a current and executable application image, unless the application image is confirmed as being completely written to a memory resource within the target device and also successfully performs a validation function. During the writing and validation steps of a "new" or second set of instructions in the form of a second application image, an "old" or first set of instructions in the form of a first application image is not overwritten, and maintained executable, so as to ensure that the target device is not rendered unbootable as a result of the loading of the second set of instructions. The invention is advantageous in that the frequency of service calls by personnel to target devices, which may comprise embedded devices in inaccessible locations, may be reduced.

Returning to FIG. **2**, the processor **42**, EEPROM **44**, and RAM **48** are each shown to include a sequence of machine-readable instructions **160**, or at least a portion of such a sequence of instructions, that when executed by the machine, such as for example the modem **16**, cause the machine to perform any one of the steps described above in the specification, and specifically the steps described with reference to the flow charts shown above. Accordingly, for the purposes of the specification, the term "machine-readable medium" shall be taken to include any medium that is capable of storing or otherwise accommodating a sequence of instructions for execution by a machine, and that cause the machine to perform the methodologies of the present invention. Accordingly, the term "machine-readable medium" shall be taken to include, but not be limited to, solid-state memories, optical and magnetic disks, and carrier-wave signals. For example, the instructions for performing the methodologies of the present invention may be propagated to a target machine via a communication link and be encoded within a suitable carrier-wave signal.

Thus, a method and apparatus for installing a second set of instructions within a machine, the machine including a memory resource storing first and second set of instructions, have been described. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader scope and spirit of the invention. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method of installing a second set of instructions within a machine, the machine including a memory resource storing a first set of instructions, the method including:

executing the first set of instructions;

loading a second set of instructions into the memory resource from a remote device, while maintaining the first set of instructions within the memory resource;

validating the second set of instructions, the validating including determining whether the machine, operating under the direction of the second set of instructions, successfully performs a predetermined function; and

if the second set of instructions is valid, indicating the second set of instructions as executable in place of the first set of instructions,

wherein the predetermined function includes establishing communications between the machine and the remote device under direction of the second set of instructions.

2. The method of claim **1** wherein the validating includes determining whether the second set of instructions is completely stored within the memory resource.

3. The method of claim **2** wherein the validating includes, subsequent to the determination of whether the second set of instructions is completely stored within the memory resource, resetting the machine for execution of the second set of instructions.

4. The method of claim **1** wherein the validating includes determining whether execution of the second set of instructions has been previously attempted.

5. The method of claim **1** wherein the machine is a modem, and wherein the validating includes determining whether the modem establishes a communication link with the remote device.

6. The method of claim **5** wherein establishing of the communication link includes timing out a communication operation after a predetermined time threshold.

7. The method of claim 1 including, subsequent to the performance of the predetermined function, resetting in the machine for execution of the second set of instructions.

8. The method of claim 1 including loading the second set of instructions into the memory resource subsequent to commencement of the execution of the first set of instructions, while maintaining the first set of instructions within the memory resource.

9. The method of claim 8 wherein the loading includes loading an object code image into a read-only memory.

10. The method of claim 9 wherein the loading of the object code image includes loading a compressed object code image.

11. The method of claim 8 wherein the loading includes downloading the second set of instructions to the memory resource from a remote location.

12. The method of claim 11 wherein the executing of the second set of instructions includes decompressing an object code image and transferring the object code image from a read-only memory to a random access memory.

13. The method of claim 1 wherein the indicating includes setting a pointer to indicate the second set of instructions for default execution.

14. The method of claim 1 including resetting the machine, and executing the second set of instructions.

15. A machine comprising:

a memory resource to store a first and second set of instructions; and

logic to execute the first set of instructions, to load the second set of instructions into the memory resource from a remote device while maintaining the first set of instructions within the memory resource, to validate the second set of instructions by determining whether the machine, when executing the second set of instructions, successfully performs a predetermined function, and to indicate the second set of instructions as executable in place of the first set of instructions,

wherein the predetermined function includes establishing communications between the machine and the remote device under direction of the second set of instructions.

16. The machine of claim 15 wherein the logic determines whether the second set of instructions was completely written to the memory resource of the machine.

17. The machine of claim 15 wherein the logic determines whether execution of the second set of instructions has previously been attempted.

18. The machine of claim 15 wherein the logic determines whether a communication link is established between the machine and a remote device.

19. The machine of claim 18 wherein the logic determines whether the communication link is established within a predetermined time-out period.

20. The machine of claim 15 including an interface via which the second set of instructions is loaded into the memory resource subsequent to the commencement of execution of the first set of instructions, while maintaining the first set of instructions within the memory resource.

21. The machine of claim 20 wherein the interface facilitates the loading of the second set of instructions into the memory resource from the remote location.

22. The machine of claim 15 wherein the memory resource comprises a read-only memory, and the second set of instructions comprises an object code image.

23. The machine of claim 15 including a pointer to indicate either the first or the second set of instructions as a default set of instructions for execution by the machine.

24. The machine of claim 23 wherein the logic sets the pointer to indicate the second set of instructions for default execution after validating the second set of instructions.

25. The machine of claim 24 wherein the pointer comprises a memory resource storing a flag.

26. A machine-readable medium storing a sequence of instructions that, when executed by a machine, cause the machine to:

load a replacement set of instructions from a remote device, while executing an existing set of instructions;

validate the replacement set of instructions by determining whether the replacement set of instructions executes to establish communications with the remote device; and

if the second set of instructions is valid, then indicate the second set of instructions as executable by the machine in place of the first set of instructions.

27. A method to install replacement software, the method including:

downloading the replacement software from a provider to a target device, the replacement software to replace existing software stored by the target device; and

causing execution of the replacement software by the target device so as to validate the replacement software,

wherein the replacement software is validated upon detection of a successful establishment of communications between the target device, operating under direction of the replacement software, and the provider.

28. The method of claim 27 including downloading the replacement software to the target device subsequent to commencement of execution of the existing software, while maintaining the existing software at the target device.

29. The method of claim 27 wherein the downloading of the replacement software includes downloading an object code image into read-only memory of the target device.

30. The method of claim 29 wherein the downloading of the object code image includes downloading a compressed code image.

31. The method of claim 29 wherein the causing of the execution of the replacement software includes decompressing the object code image, and transferring the object code image from the read-only memory to a random access memory.

32. A machine-readable medium storing a sequence of instructions that, when executed by a source machine, cause the source machine to perform the operations of:

downloading replacement software from a provider to a target device, the replacement software to replace existing software stored by the target device; and

causing execution of the replacement software by the target device so as to validate the replacement software,

wherein the replacement software is validated upon detection of a successful establishment of communications between the target device, operating under the direction of the replacement software, and the provider.

* * * * *

US005881236A

United States Patent [19]

Dickey

[11] Patent Number: 5,881,236

[45] Date of Patent: Mar. 9, 1999

[54] **SYSTEM FOR INSTALLATION OF SOFTWARE ON A REMOTE COMPUTER SYSTEM OVER A NETWORK USING CHECKSUMS AND PASSWORD PROTECTION**

[75] Inventor: **Conwell J. Dickey**, Ft. Collins, Colo.

[73] Assignee: **Hewlett-Packard Company**, Palo Alto, Calif.

[21] Appl. No.: **639,160**

[22] Filed: **Apr. 26, 1996**

[51] **Int. Cl.**[6] **G06F 9/445**; G06F 15/177

[52] **U.S. Cl.** **395/200.51**; 395/200.5; 395/200.52; 395/712

[58] **Field of Search** 395/200.5, 200.51, 395/200.52, 712

[56] **References Cited**

U.S. PATENT DOCUMENTS

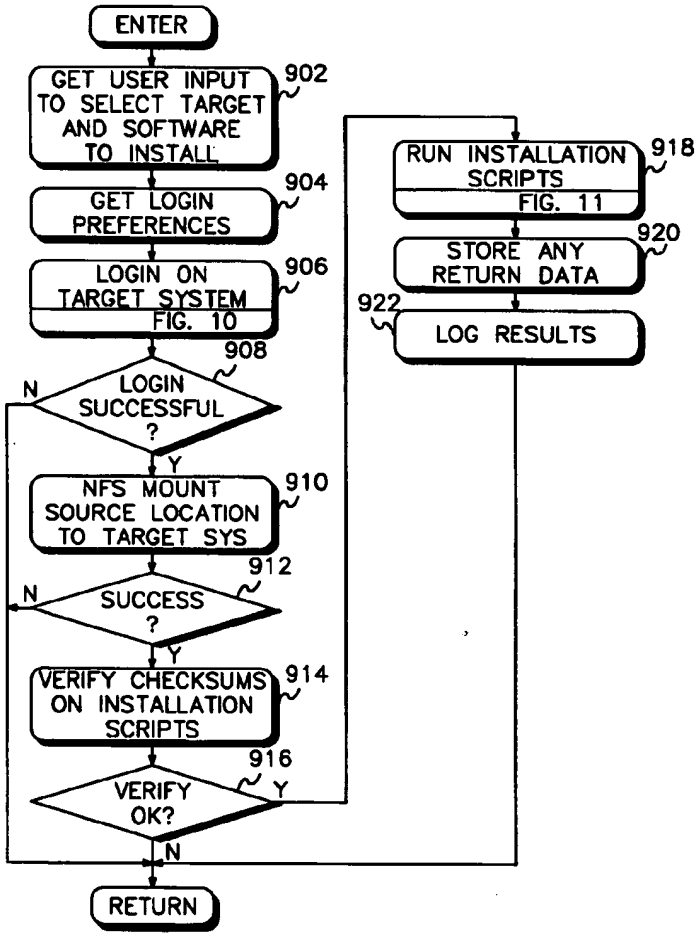
5,008,814	4/1991	Mathur	395/200.51
5,060,263	10/1991	Bosen et al.	380/25
5,421,009	5/1995	Platt	395/200.51
5,550,968	8/1996	Miller et al.	345/332
5,555,416	9/1996	Owens et al.	395/712
5,598,536	1/1997	Slaughter, III et al.	395/200.49

Primary Examiner—Daniel H. Pan
Assistant Examiner—Jeffrey Rossi

[57] **ABSTRACT**

A system to perform software on a remote computer system automatically over a network using a local computer system. The software being performed is placed on a storage device connected to the local computer. The local computer executes software that logs on to the remote computer as if the local computer were a remote terminal and sends a command to the remote computer to cause it to connect to the storage device through the network. The local computer then sends commands to the remote computer to cause the remote computer to perform the software located on the storage device. Each software program run on the remote computer examines the remote computer to determine the type of software that is running on the remote computer and selects the software being executed to be compatible with the software on the remote computer.

10 Claims, 11 Drawing Sheets



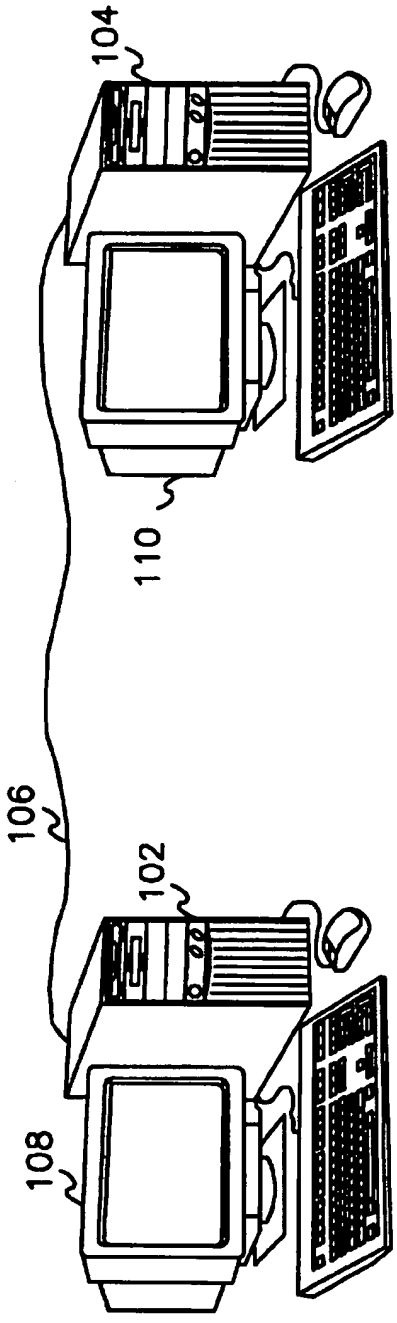


FIG. 1

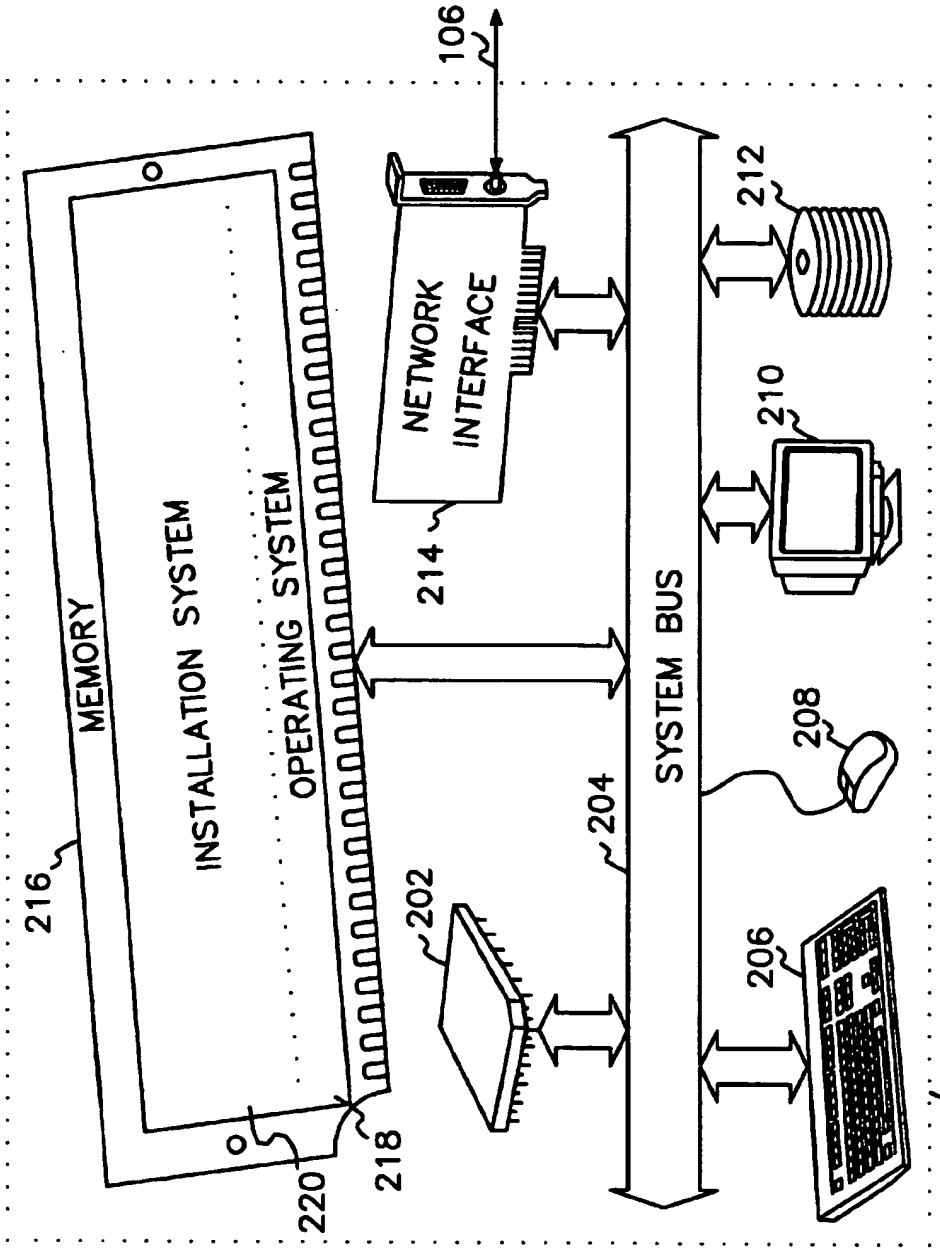

106
NETWORK INTERFACE
214
MEMORY
INSTALLATION SYSTEM
OPERATING SYSTEM
216
218
220
SYSTEM BUS
204
202
212
210
208
206
102

FIG. 2

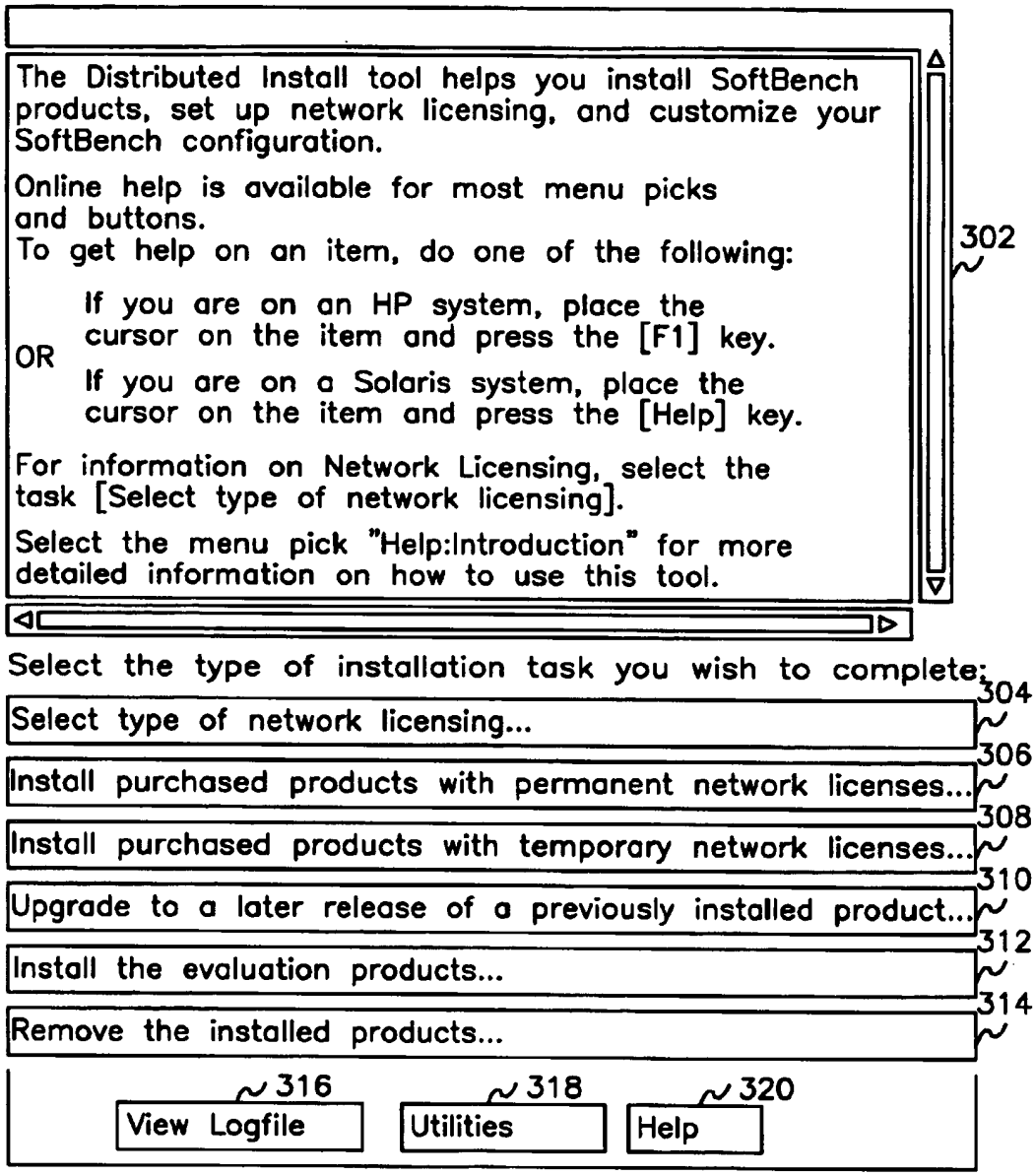

The Distributed Install tool helps you install SoftBench products, set up network licensing, and customize your SoftBench configuration.
Online help is available for most menu picks and buttons.
To get help on an item, do one of the following:
If you are on an HP system, place the cursor on the item and press the [F1] key.
OR
If you are on a Solaris system, place the cursor on the item and press the [Help] key.
For information on Network Licensing, select the task [Select type of network licensing].
Select the menu pick "Help:Introduction" for more detailed information on how to use this tool.
302
Select the type of installation task you wish to complete:
Select type of network licensing...
304
Install purchased products with permanent network licenses...
306
Install purchased products with temporary network licenses...
308
Upgrade to a later release of a previously installed product...
310
Install the evaluation products...
312
Remove the installed products...
314
View Logfile
316
Utilities
318
Help
320

FIG. 3

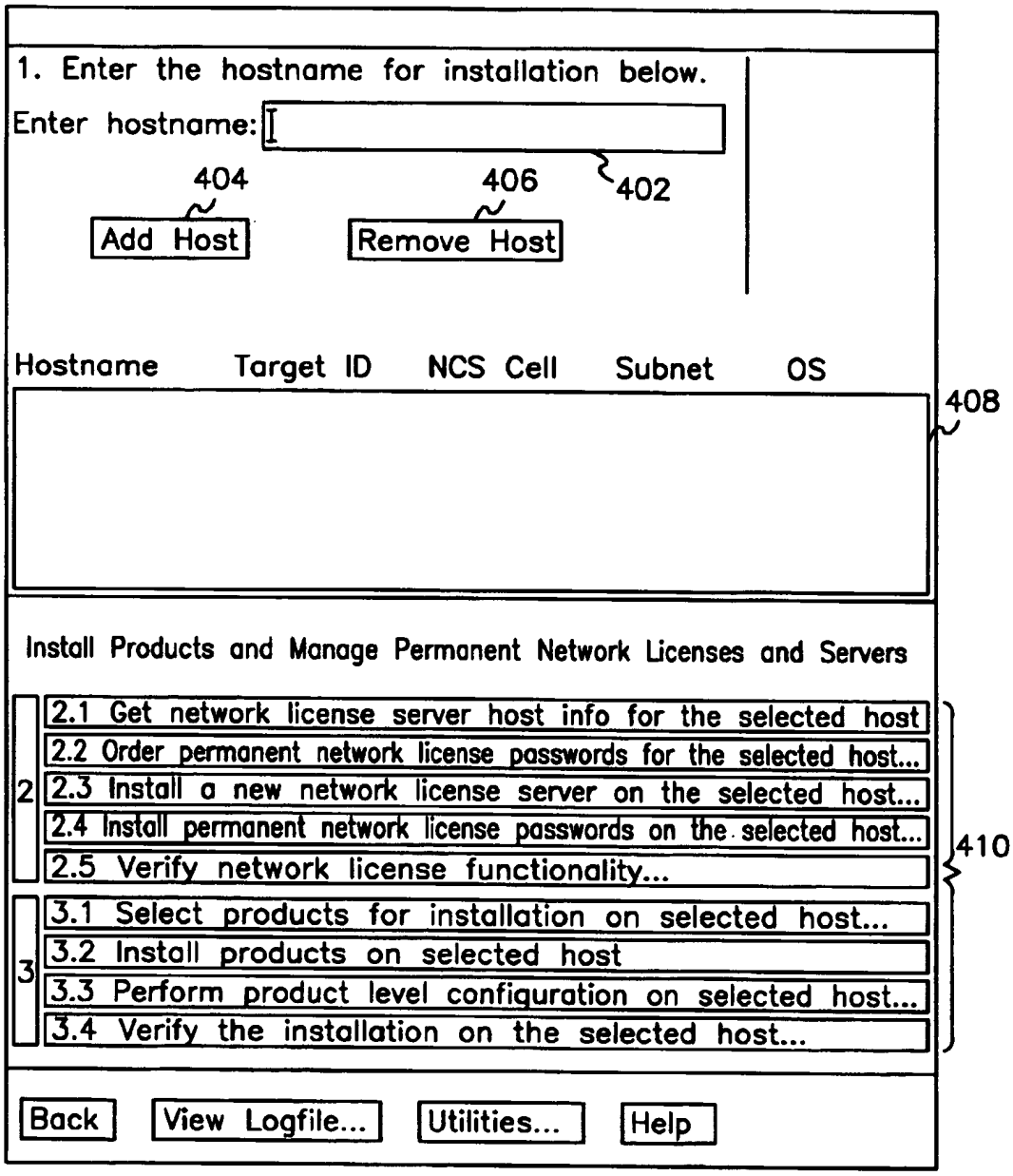

1. Enter the hostname for installation below.
Enter hostname:
404
406
402
Add Host
Remove Host
Hostname
Target ID
NCS Cell
Subnet
OS
408
Install Products and Manage Permanent Network Licenses and Servers
2
2.1 Get network license server host info for the selected host
2.2 Order permanent network license passwords for the selected host...
2.3 Install a new network license server on the selected host...
2.4 Install permanent network license passwords on the selected host...
2.5 Verify network license functionality...
3
3.1 Select products for installation on selected host...
3.2 Install products on selected host
3.3 Perform product level configuration on selected host...
3.4 Verify the installation on the selected host...
410
Back
View Logfile...
Utilities...
Help

FIG. 4

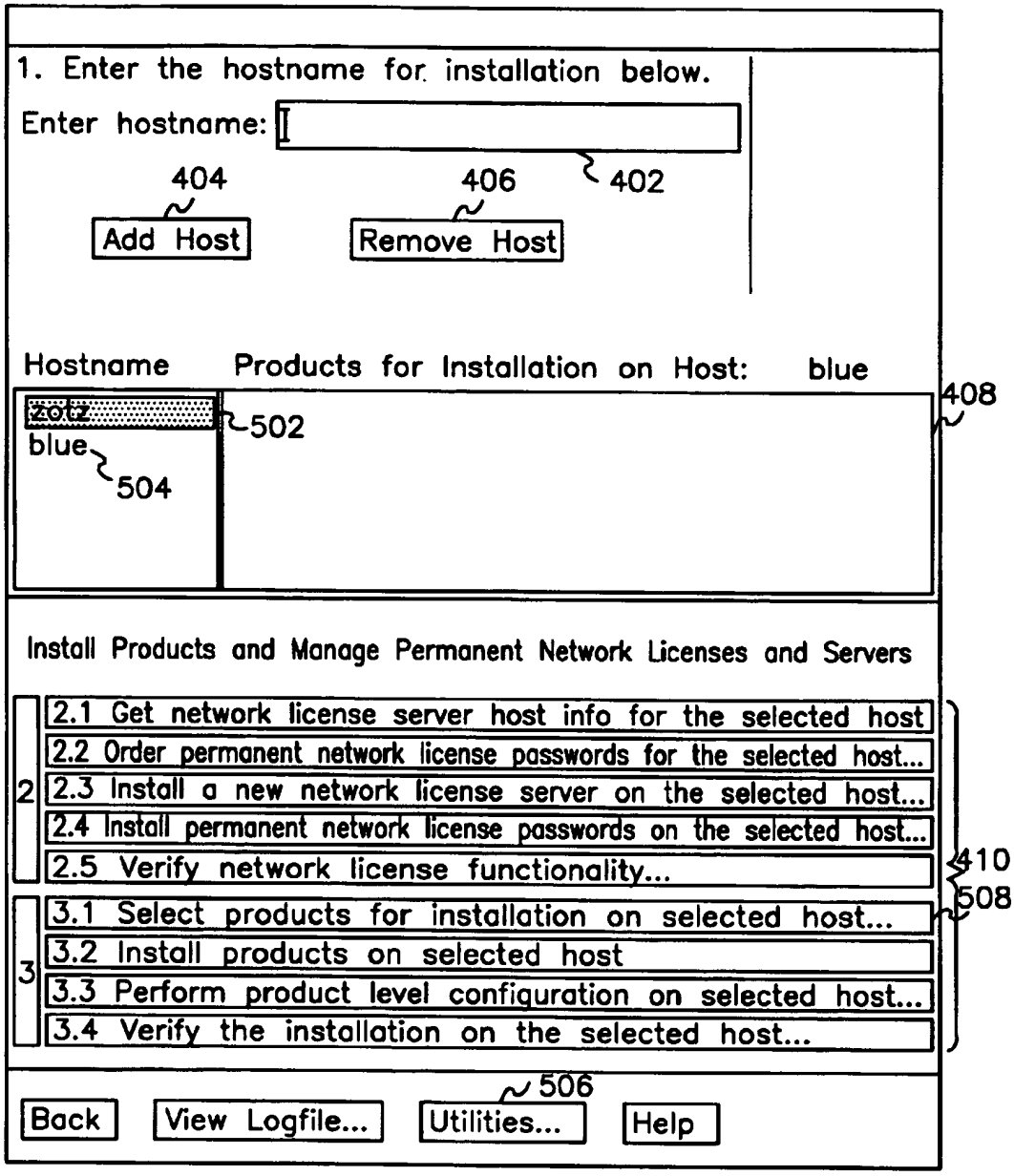

1. Enter the hostname for installation below.
Enter hostname:
404
406
402
Add Host
Remove Host
Hostname
Products for Installation on Host: blue
408
zotz
502
blue
504
Install Products and Manage Permanent Network Licenses and Servers
2
2.1 Get network license server host info for the selected host
2.2 Order permanent network license passwords for the selected host...
2.3 Install a new network license server on the selected host...
2.4 Install permanent network license passwords on the selected host...
2.5 Verify network license functionality...
410
3
3.1 Select products for installation on selected host...
508
3.2 Install products on selected host
3.3 Perform product level configuration on selected host...
3.4 Verify the installation on the selected host...
506
Back
View Logfile...
Utilities...
Help

FIG. 5

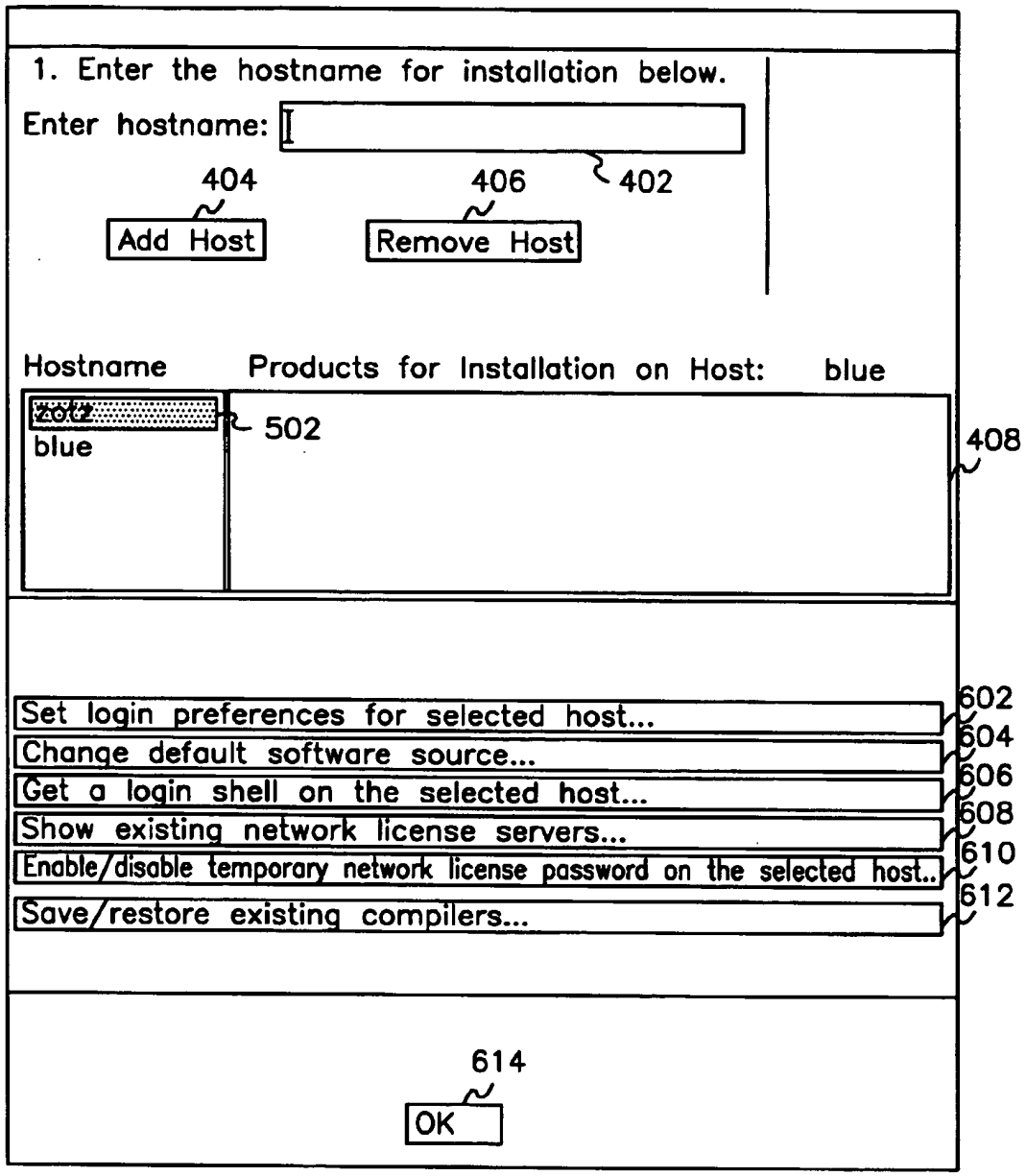

1. Enter the hostname for installation below.
Enter hostname:
402
404
Add Host
406
Remove Host
Hostname
Products for Installation on Host: blue
blue
502
408
Set login preferences for selected host...
Change default software source...
Get a login shell on the selected host...
Show existing network license servers...
Enable/disable temporary network license password on the selected host..
Save/restore existing compilers...
602
604
606
608
610
612
614
OK

FIG. 6

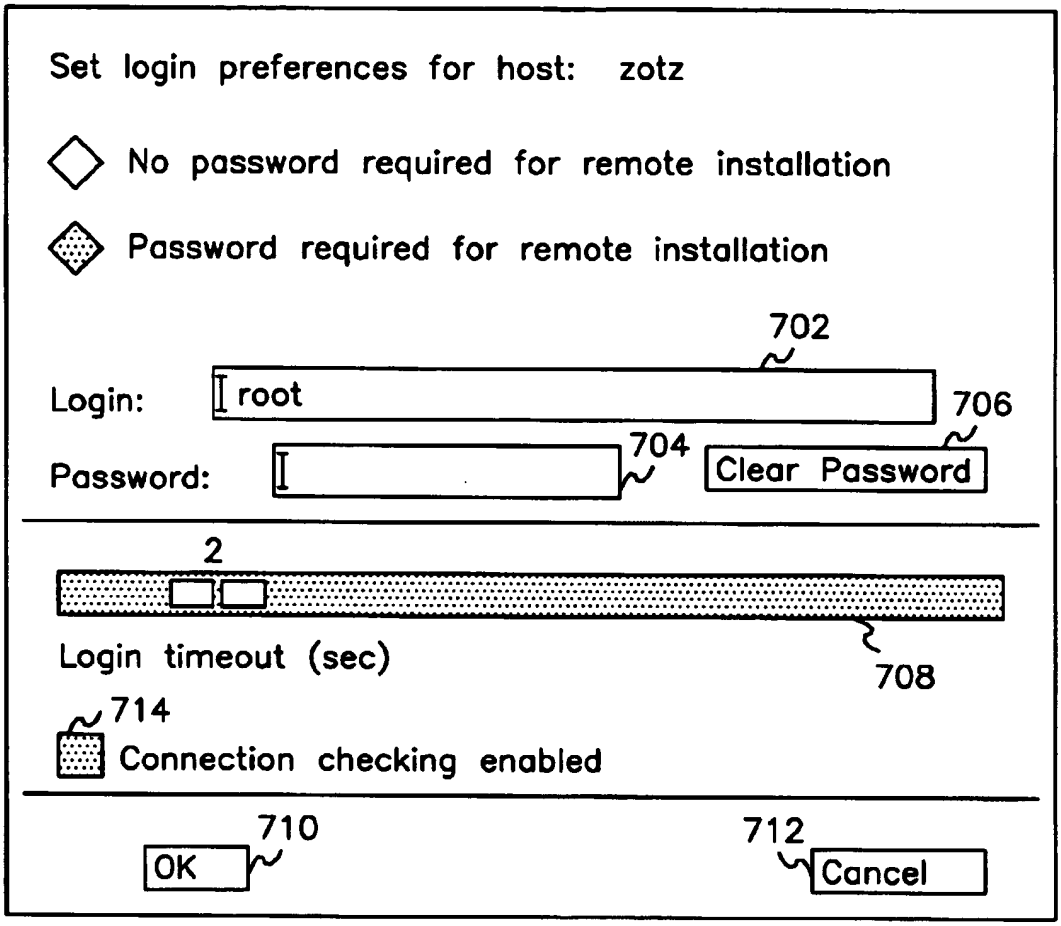

Set login preferences for host: zotz
No password required for remote installation
Password required for remote installation
702
Login:
root
706
Password:
704
Clear Password
2
Login timeout (sec)
708
714
Connection checking enabled
710
OK
712
Cancel

FIG. 7

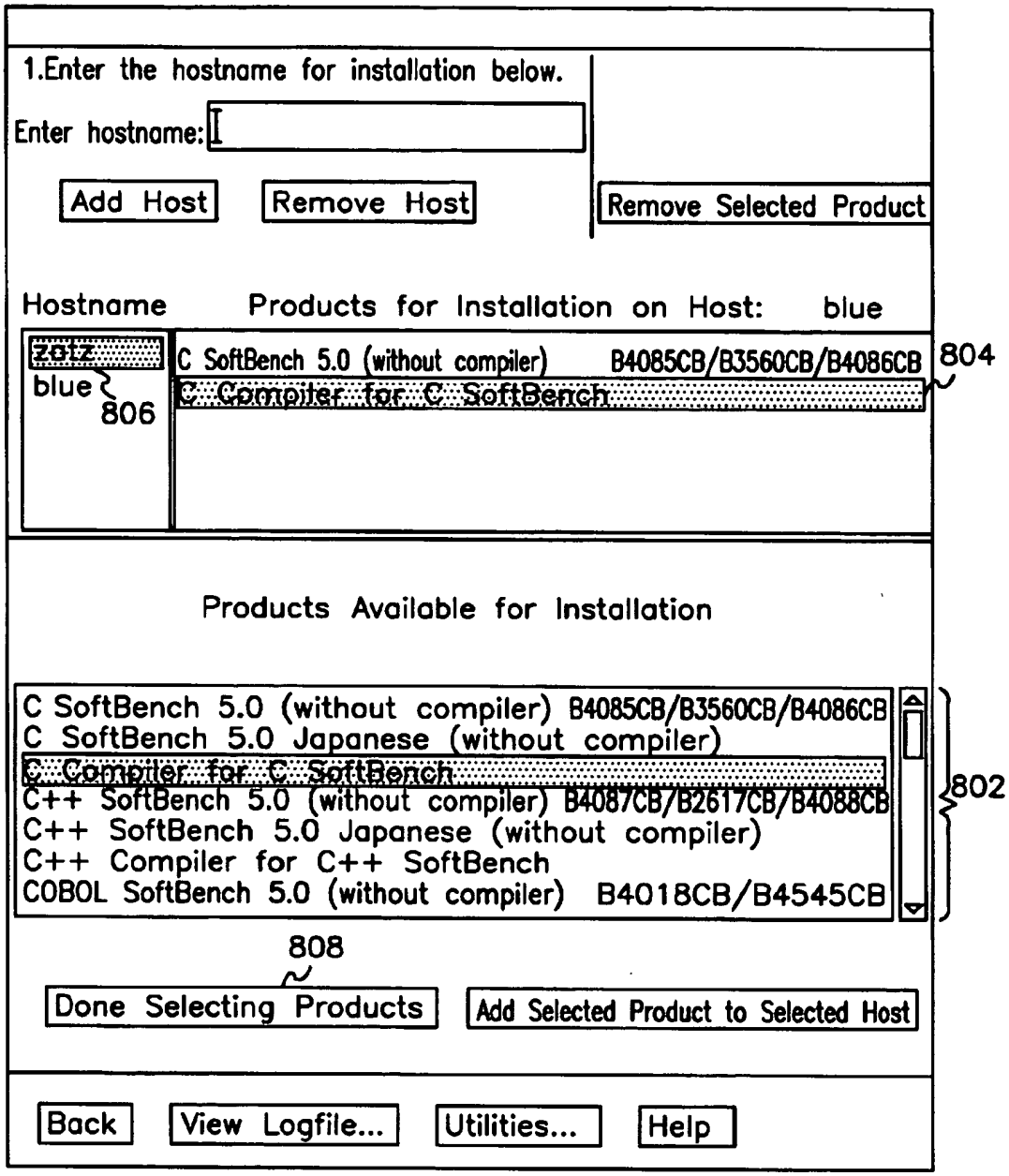

1.Enter the hostname for installation below.
Enter hostname:
Add Host
Remove Host
Remove Selected Product
Hostname
Products for Installation on Host: blue
zatz
blue
806
C SoftBench 5.0 (without compiler) B4085CB/B3560CB/B4086CB
C Compiler for C SoftBench
804
Products Available for Installation
C SoftBench 5.0 (without compiler) B4085CB/B3560CB/B4086CB
C SoftBench 5.0 Japanese (without compiler)
C Compiler for C SoftBench
C++ SoftBench 5.0 (without compiler) B4087CB/B2617CB/B4088CB
C++ SoftBench 5.0 Japanese (without compiler)
C++ Compiler for C++ SoftBench
COBOL SoftBench 5.0 (without compiler) B4018CB/B4545CB
802
808
Done Selecting Products
Add Selected Product to Selected Host
Back
View Logfile...
Utilities...
Help

FIG. 8

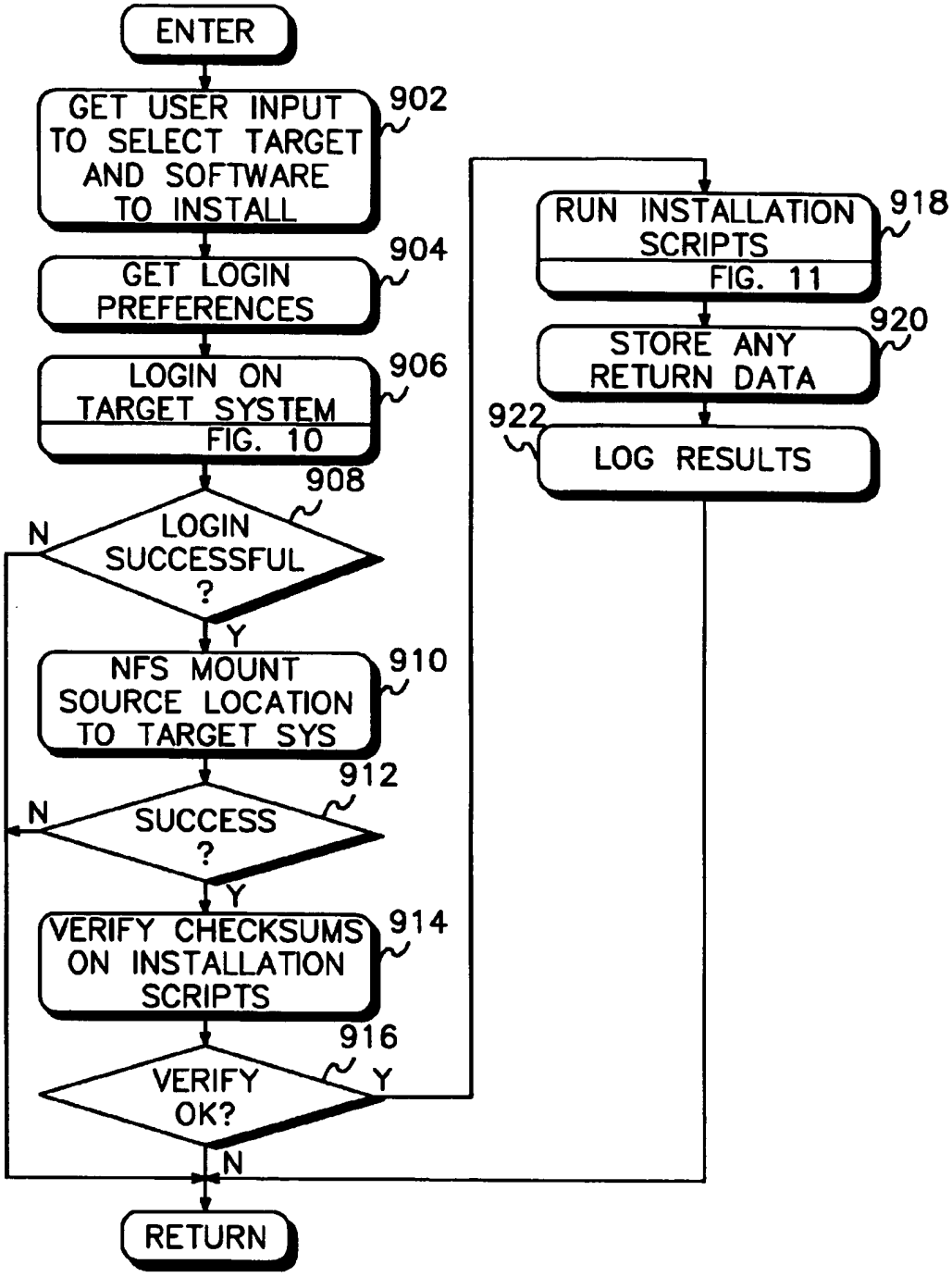

ENTER
GET USER INPUT TO SELECT TARGET AND SOFTWARE TO INSTALL
902
GET LOGIN PREFERENCES
904
LOGIN ON TARGET SYSTEM
FIG. 10
906
908
LOGIN SUCCESSFUL ?
N
Y
NFS MOUNT SOURCE LOCATION TO TARGET SYS
910
912
SUCCESS ?
N
Y
VERIFY CHECKSUMS ON INSTALLATION SCRIPTS
914
916
VERIFY OK?
Y
N
RUN INSTALLATION SCRIPTS
FIG. 11
918
STORE ANY RETURN DATA
920
922
LOG RESULTS
RETURN

FIG. 9

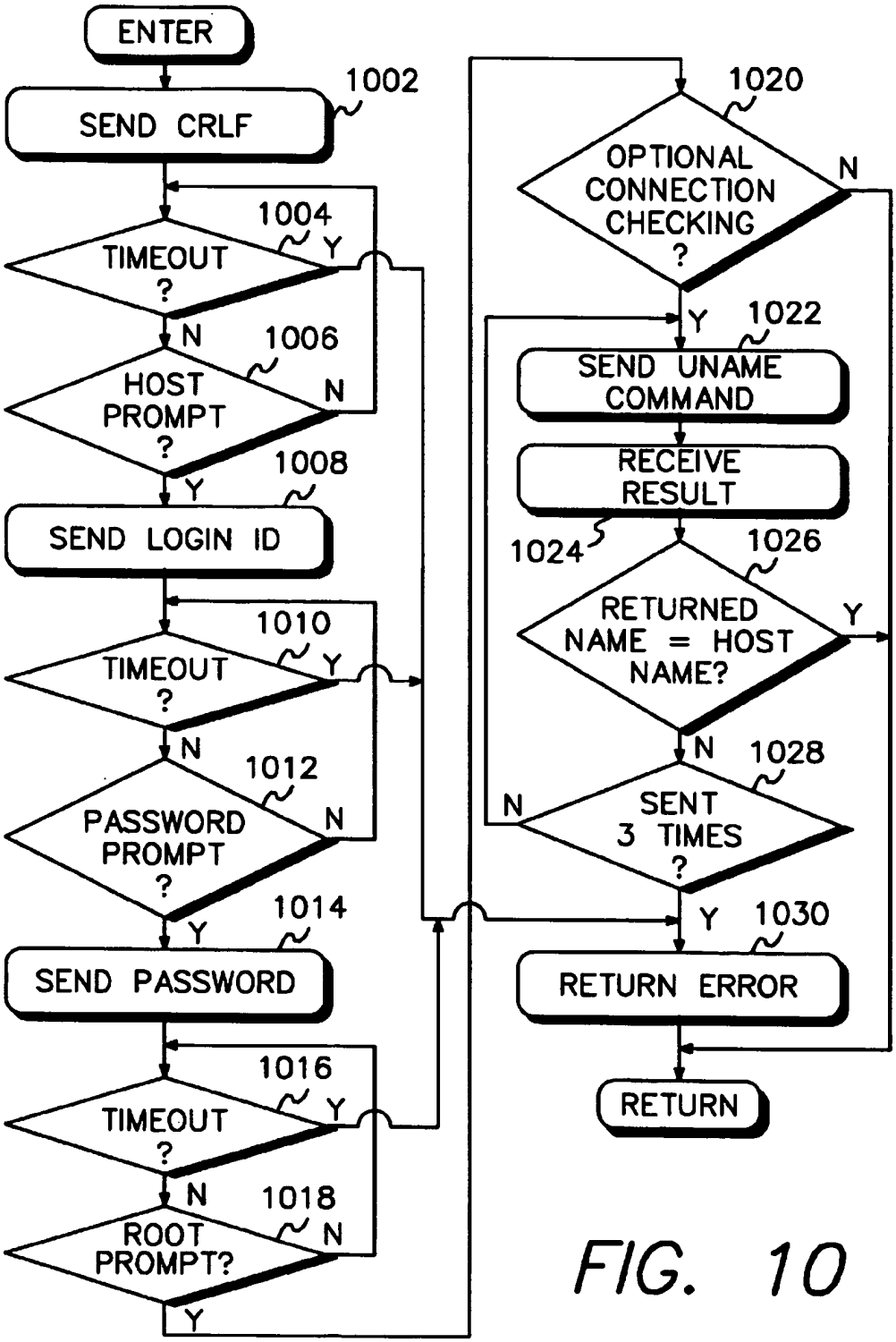

ENTER
1002
SEND CRLF
1004
TIMEOUT ?
Y
N
1006
HOST PROMPT ?
N
Y
1008
SEND LOGIN ID
1010
TIMEOUT ?
Y
N
1012
PASSWORD PROMPT ?
N
Y
1014
SEND PASSWORD
1016
TIMEOUT ?
Y
N
1018
ROOT PROMPT?
N
Y
1020
OPTIONAL CONNECTION CHECKING ?
N
Y
1022
SEND UNAME COMMAND
RECEIVE RESULT
1024
1026
RETURNED NAME = HOST NAME?
Y
N
1028
SENT 3 TIMES ?
N
Y
1030
RETURN ERROR
RETURN

FIG. 10

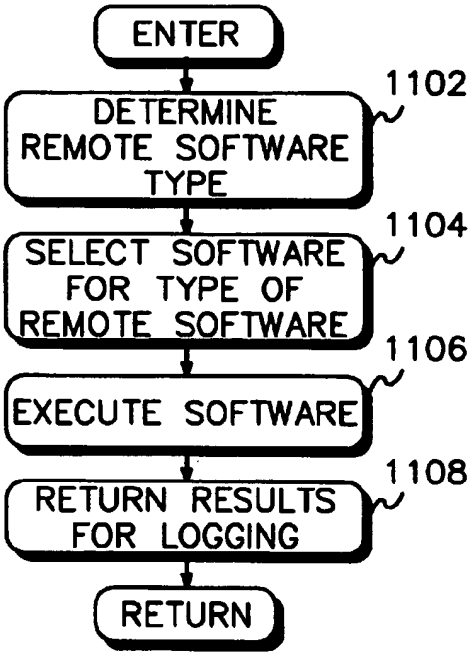

ENTER
DETERMINE REMOTE SOFTWARE TYPE
1102
SELECT SOFTWARE FOR TYPE OF REMOTE SOFTWARE
1104
EXECUTE SOFTWARE
1106
RETURN RESULTS FOR LOGGING
1108
RETURN

FIG. 11

SYSTEM FOR INSTALLATION OF SOFTWARE ON A REMOTE COMPUTER SYSTEM OVER A NETWORK USING CHECKSUMS AND PASSWORD PROTECTION

FIELD OF THE INVENTION

This invention relates to computer systems and more particularly to software installation within computer systems. Even more particularly, the invention relates to remotely installing software through a network on a computer system.

BACKGROUND OF THE INVENTION

When software utilizing a client-server architecture is being distributed to customers, the server portion of the software is installed on a server computer system, and the client portion of the software is typically installed on each client computer system attached to the network. The installation of the server portion of the software is routine, since usually there is only one, or a very few, server computer systems. Installation of the client portion of the software is considerably more difficult.

Often there are many client computer systems, sometimes hundreds, and the client portion of the software must be installed individually on each machine. One prior art method of accomplishing this installation is for an installer to manually take the software distribution media to each client machine and individually install from the media onto that machine. A more efficient method is to first install the client portion of the software on the server machine, and then an installer goes to each individual client machine, logs on to the machine, and manually enters the commands necessary to copy and install the software, from the network server, onto the client machine.

Another prior art method is to manually place a download program on the server and client machines, then the download program can be used to automatically install other client-server software over a network connecting the server and client machines. This method has the disadvantage that the download program must first be manually installed on all the client machines, which requires considerable effort when a large number of client machines are being served.

All these methods, however, require considerable time to perform the installation, either of the client server software or the download program, particularly where a large number of client computer systems are involved.

In addition to installation, these same problems exist for reconfiguring the client computer systems, and also for uninstalling the client computer systems, as well as many other operations that may have to be done on the client computer systems.

There is need in the art then for a way of installing the client portion of client server software onto client machines without requiring that the software or a download program be manually copied to each client computer system. The present invention meets this and other needs in the art.

SUMMARY OF THE INVENTION

It is an aspect of the present invention to provide a method of remotely installing software or data files over a network onto a remote computer system.

Another aspect of the invention is to remotely install software without first installing a download program on the remote computer system.

It is another aspect of the invention to use remote terminal log on facilities and network file services facilities of the target computer to accomplish the installation.

A still further aspect of the invention is to perform other software remotely on a remote computer system.

The above and other aspects of the invention are accomplished in a method that allows installation of software, or other software operations, on a remote computer system to be done automatically over a network using a local computer system. The software to be installed is placed on a storage device connected to the local computer system, for example a CDROM device. The local computer system executes local software that logs on to the remote computer system as if the local computer system were a terminal of the remote computer system. The local computer system then sends a command to the remote computer system to cause it to connect to the storage device through the network, typically through Network File Services (NFS).

The local computer system then validates the software stored on the storage device by sending a command to the remote computer system to cause the remote computer system to perform a checksum test of the software. When the remote computer system "displays" the checksums to the local computer system, the local software verifies that the checksums are correct by comparing them to checksum values stored in the local software.

The local software then sends commands to the remote computer system to cause the remote computer system to perform the software located on the storage device, which installs the software onto the remote computer system, or reconfigures the remote computer system, or uninstalls software from the remote computer system, or performs any other function within the remote computer system.

When the software is to be run on the remote computer system, the software in the local computer system first examines the remote computer system to determine the type of operating system software that is running on the remote computer system. Once the operating system is determined, the software on the local computer system then selects software to be run on the remote computer system to be compatible with the operating system of the remote computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other aspects, features, and advantages of the invention will be better understood by reading the following more particular description of the invention, presented in conjunction with the following drawings, wherein:

FIG. 1 shows a diagram of a local and a remote computer system connected by a computer network and illustrates the remote installation process;

FIG. 2 shows a block diagram of the local computer system that performs the remote install;

FIG. 3 shows a initial screen presented to the user on the local computer system;

FIG. 4 shows the screen presented to the user on the local computer system when installing software with permanent network licenses;

FIG. 5 shows the screen presented to the user on the local computer system after the user has selected one or more remote computer systems;

FIG. 6 shows the screen presented to the user on the local computer system after the user has selected the utilities button;

FIG. 7 shows the screen presented to the user on the local computer system to set the login id and password for use with the remote computer system;

FIG. 8 shows the screen presented to the user on the local computer system to allow the user to select the software to be installed;

FIG. 9 shows a flowchart of the local computer system software process;

FIG. 10 shows a flowchart of the local computer software process of logging on to the remote computer system and verifying the type of operating system running on the remote computer system; and

FIG. 11 shows a flowchart of the process of executing scripts on the remote computer system to perform the installation or other desired functions.

DESCRIPTION OF THE PREFERRED EMBODIMENT

The following description is of the best presently contemplated mode of carrying out the present invention. This description is not to be taken in a limiting sense but is made merely for the purpose of describing the general principles of the invention. The scope of the invention should be determined by referencing the appended claims.

FIG. 1 shows a diagram of a computer network having two computer systems. Referring now to FIG. 1, a local computer system **102** has a display element **108**. The local computer system **102** is connected to a remote computer system **104** over a network **106**. The network **106** is typically a local area network (LAN), but may be any type of network, including serial interfaces using modems. The present invention comprises software running in the local computer system **102** and also in the remote computer system **104** to install software, uninstall software, or perform other software functions from the local computer system **102** over the network **106** to the remote computer system **104**.

FIG. 2 shows a block diagram of the local computer system **102**. This same block diagram could also be applicable to the remote computer system **104**. Referring now to FIG. 2, the computer system **102** contains a processing element **202** which communicates to other elements of the computer system **102** over a system bus **204**. A keyboard **206** allows text input to the computer system **102** and a mouse **208** allows graphical locator input to the computer system **102**. A graphics display **210** provides for graphics and text output to be viewed by a user of the computer system **102** and a disk **212** stores the software and data of the present invention, as well as an operating system and other user data of the computer system **102**. A network interface **214** allows the computer system **102** to transfer data and commands over the network **106** (FIG. 1).

A memory **216** contains and operating system **218**, which is typically the Unix operating system. Those skilled in the art will recognize that other operating systems could also work with the present invention. Memory **216** also contains the installation software **220** which comprises the present invention. For the local computer system, the installation software **220** is the local software. For the remote computer system, the installation software **220** is the installation programs, or other software to be executed remotely, from the storage device.

For the remote computer system, disk **212** is used to store the software being installed. For the local computer system, disk **212** contains the software that will be transferred over the network **106** through network interface **214**.

FIG. 3 shows the initial screen presented to the user when the installation software is first loaded onto the local computer system. Referring now to FIG. 3, area **302** shows introductory text for the user. Below this section are a series of command lines starting with block **304** which allows the user to transfer to a screen to select the type of network licensing, area **306** allows the user to install purchase products with permanent network licenses, area **308** allows the user to install products with temporary network licenses, area **310** allows the user to upgrade to a later release, area **312** allows the user to install evaluation products, and area **314** allows the user to remove installed products. Areas **304–314** operate as buttons so that when the user clicks one of these the system transfers to another screen.

Below these sections are three buttons that allow the user to perform various functions, button **316** allows the user to view the logfile that has been created from previous installations, block **318** allows the user to access a series of utilities, which are shown below with respect to FIG. 6, and button **320** allows the user to request help.

If the user presses button **306**, the system displays the screen of FIG. 4. Referring to FIG. 4, area **402** allows the user to enter the host name of the remote computer system on which the software will be installed, and after entering a name in the area **402**, button **404** allows the user to add this name to the area **408**. Once a host name is within area **408**, block **406** allows the user to remove that name. After a host name is added to the area **408**, the buttons defined in area **410** become active allowing the user to select additional options.

FIG. 5 shows the screen of FIG. 4 after some hosts have been selected. Referring now to FIG. 5, two host names, one in area **502** and one in area **504** have been added to the list of host names by the user of the system. In FIG. 5, the host name "zotz" in area **502** is highlighted such that if the user were to click the remove host button **406** this host would be removed. Furthermore, if the user clicks any of the buttons in area **410**, the system will perform the requested function for the host name highlighted in the area **502**.

When the user presses the utilities **506**, the screen of FIG. 6 is displayed. Referring now to FIG. 6, with the host name "zotz" highlighted in area **502**, the user can press any of the buttons **602–612** to perform one of the utility functions for this host. After all desired utility functions have been performed, the user clicks the OK button **614** to return to the screen of FIG. 5.

When the user clicks button **602** to set the login preferences, the system displays the screen of FIG. 7. Referring now to FIG. 7, this screen allows the user to enter the host name for the remote computer system in area **702**, and allows the user to enter the password for the host name of **702** into the area **704**. When the user enters the host name for the remote computer system in area **708**, the name is displayed as shown in FIG. 7 where the logon ID for this host is "root". When the user enters the password for this logon ID into area **704**, however, the password is not displayed so that the password remains secret. Furthermore, the number of characters in the password is not displayed, to preserve the length of the password in secret, thus the cursor within the area **704** always remains at the left edge of the field. This creates a problem, however, should the user make a typing error while entering the password. If the user desires to backspace over the password that has been entered, the user has no way of knowing how far to backspace. To resolve this problem, the clear password button **806** is disabled until at lease one character has been typed into the password field **704**. After one or more characters have been typed into the password field **704**, the clear password button **706** is enabled and the text of the

button is displayed in normal lettering, to indicate to the user that at least one character has been typed in the password. Should the user backspace and remove characters from the password field **704**, the clear password button **706** will remain enabled until all characters have been removed from the password field **704**. Once all characters have been removed from the password field **704**, the clear password button **706** will again be disabled and the button text will be displayed in a grayed (light lettering) format. This feature allows a password to be entered without revealing either the characters of the password or the length of the password, while still providing a way for user easily to correct errors. The clear password button can also be used to erase the entire password with a single button push.

Block **708** allows the user to enter a login timeout value, which is the value used in FIG. **10** below, to determine whether the remote computer system has taken too long to respond to the login request. Once the user has entered all information on FIG. **7**, the user clicks the OK button **710** to save this information or clicks the Cancel button **712** to erase all the information just entered. In either case, FIG. **7** returns to FIG. **6**.

The user may select products to be entered by pressing the select products button **508** on FIG. **5**, at which time the system will display the screen of FIG. **8**. Referring now to FIG. **8**, area **802** displays all the products that are available for installation, and once one of these products has been selected, it is removed to the area **804** to be installed on the remote computer system that is highlighted in area **806**. Once the user has selected the desired products, the user clicks the done selecting products button **808** which causes the system to enter the flowchart of FIG. **9** to perform the installation of the selected products.

FIG. **9** shows a flowchart of the installation process. Referring now to FIG. **9**, after entry, block **302** gets user input to select the remote computer system and to select the software that will be installed on the remote computer system. This user input was described above. Block **904** gets the login preferences for the remote computer system, which was described above with respect to FIG. **7**. These preferences include the login ID to be used and the password associated with that ID. Block **906** then calls FIG. **10** to perform the login to the target system. If the login is successful, block **908** transfers to block **910** which sends a command to the target system to cause the target system to mount the installation device located on the local computer system as a network file service (NFS) device. Using NFS, the disk located on the local computer system appears to the remote computer system as if the disk is located there. Because of this feature, the remote computer system can execute programs contained on the installation storage device, and easily transfer data files and software files from the installation storage device to another storage device located within the remote computer system.

If the NFS mount is successful, block **912** goes to block **914** which sends a command to the remote computer system to verify the checksums on all of the scripts to be executed, also called programs, contained on the installation storage device. This checksum is used to insure that the scripts have not been changed. When the remote computer system verifies the checksums it displays the result, and this display is sent back to the local computer system which takes the checksums and compares them to checksums within the local computer system software. If the checksums are not correct, or if the login or NFS mounts have failed, control returns without performing the scripts. By keeping the checksum values within the local computer system software, and verifying them before executing the script software, the system prevents any changes to the scripts being executed on the remote computer system, unless the local computer system software is also updated, thus preventing unauthorized changes to the software being installed.

If the checksums are correct, block **916** transfers to block **918** which calls FIG. **11** to run the scripts. After the scripts have been run, block **920** stores any data returned by the script, and block **922** writes a log file showing the results of the login process. This log file shows the commands sent to the remote computer system and the information sent back by the remote computer system in response to each command. This log file helps in troubleshooting problems with the system, and also supplies a record of the activity.

FIG. **10** shows a flowchart of the process of logging on to the remote computer system, called from block **906** of FIG. **9**. Referring now to FIG. **10**, after entry, block **1002** sends a carriage return followed by a line feed character to the remote computer system, and then block **1004** determines whether a timeout has occurred. If no timeout has occurred, block **1004** goes to block **1006** which determines whether a host prompt has been received from the remote computer system. If no host prompt has yet been received, block **1006** returns to block **1004** to loop until either a host prompt is received or a timeout is received. If a timeout is received, block **1004** displays an error and returns to FIG. **9**.

Once a host prompt is received, control goes to block **1008** which sends the login ID that was entered on FIG. **7**. Block **1010** then determines whether a timeout has occurred, and if not goes to block **1012** to determine whether a prompt for a password has been received. If the password prompt has not yet been received, block **1012** returns to block **1010** which loops until either a timeout occurs or a password prompt is received. Once a password prompt is received, control goes to block **1014** which sends the password that was entered on FIG. **7**. Block **1016** then determines whether a timeout has occurred, and if not transfers to block **1018** which looks for the prompt indicating that the login has been successful. This prompt is called the root prompt. If not, control returns to block **1016** which loops until either a timeout occurs, or the root prompt has been received. If a timeout occurs at any time, control goes to block **1032** to return an error to FIG. **9**.

Once the root prompt has been received, control goes to block **1020** which checks the Connection checking enabled box **714** (FIG. **7**), and if this box was checked by the user, control goes to block **1022**. Block **1022** sends a UNAME command to the remote computer system, and block **1024** receives the results returned by the remote computer system in response to the uname command. Block **1026** compares the results returned by the uname command to the host name **806** (FIG. **8**), and if these match, block **1026** returns to FIG. **9**. If the names do not match, control goes to block **1028** which repeats the sending of the uname command three times. If the names have not matched after three tries, block **1028** goes to block **1030** which returns an error to FIG. **9**. Those skilled in the art will recognize that many different commands could be sent instead of the uname command, in order to verify the correct remote computer system and the type of operating system being used in the remote system.

Those skilled in the art will recognize that the version, or level, or type of other software installed on the remote computer system could be determined by the method of FIG. **10**, thus allowing the scripts being executed to perform selected operations depending upon the version, level, or type of other software installed. Thus, the method of deter-

mining the type of software of FIG. **10** is not limited to operating system software.

FIG. **11** shows the run installation scripts that was called from block **918** of FIG. **9**. Referring now to FIG. **11**, block **1102** determines the type of software on the remote computer system, for example the type of operating system is running on the remote computer system. Block **1104** selects the software to be executed depending on the type of the operating system, or other software, that was determined by block **1102**. Thus, these scripts can be used to install software, or perform any other functions, on any number of operating systems, or for any one of a number of different types of software that is already installed on the remote computer system. Once the software has been selected, block **1106** executes the software on the remote computer system, and then block **1108** returns the results of the scripts for storage and logging to FIG. **9**.

Those skilled in the art will recognize that the method of the present invention is not limited to installing software on a remote computer system from a local computer system, but could be used to perform any type of operation on a remote computer system from a local computer system, for example product configuration, installing network licensing, or product removal. These other tasks would be done by performing different scripts from block **918** (FIG. **9**).

Having described a presently preferred embodiment of the present invention, those skilled in the art will appreciate that the aspects of the invention have been fully achieved, and it will be understood by those skilled in the art that many changes in construction and circuitry and widely differing embodiments and applications of the invention will suggest themselves without departing from the spirit and scope of the present invention. The disclosures and the description herein are intended to be illustrative and are not in any sense limiting of the invention, more preferably defined in scope by the following claims.

What is claimed is:

1. A computer implemented method for remotely executing software from a local computer system to a remote computer system through a computer network, the method comprising the steps of:

(a) receiving data, on the local computer system, to select the remote computer system from one of a plurality of potential remote computer systems;

(b) connecting the local computer system to the remote computer system, wherein the local computer system operates as a remote terminal of the remote computer system;

(c) connecting a storage device on the local computer system as a storage device on the remote computer system, wherein the storage device contains the software to be executed on the remote computer system;

(d) performing, within the remote computer system, a program to compute a checksum value for each software program on said storage device, wherein said program sends the checksum value to the local computer system;

(e) comparing, within the local computer system, each checksum value to a checksum value stored within the local computer system; and

(f) performing, on the remote computer system, the software stored on the storage device.

2. The method of claim **1** wherein step (a) further comprises receiving a password comprising the steps of:

(a1) displaying a screen location for entry of a password;

(a2) receiving characters of a password while not displaying any characters received;

(a3) displaying a clear password button;

(a4) disabling said clear password button until at least one character of a password has been received;

(a5) after at least one character has been received, enabling said clear password button; and

(a6) if all of said characters of said password are removed, disabling said clear password button.

3. The method of claim **1** wherein step (d) further comprises the steps of:

(d1) determining a type of software contained in said remote computer system; and

(d2) selecting a set of software, from at least two sets of software on said installation device, wherein said selected set of software matches said type of software determined in step (d1).

4. The method of claim **1** wherein the checksum stored within the local computer system is further stored within the software executing within the local computer system.

5. A computer implemented method for installing software from a local computer system to a target computer system through a computer network, the method comprising the steps of:

(a) receiving data to select the target computer system from one of a plurality of potential target computer systems;

(b) connecting the local computer system to the target computer system, wherein the local computer system operates as a remote terminal of the target computer system;

(c) connecting a storage device, containing the software to be installed, on the local computer system as a storage device on the target computer system;

(d) performing, within the target computer system, a program to compute a checksum value for each installation program on said storage device, wherein said program sends the checksum value to the local computer system;

(e) comparing, within the local computer system, each checksum value to a checksum value stored within the local computer system; and

(f) performing, on the target computer system, one or more installation programs stored on the storage device to perform the installation.

6. The method of claim **5** wherein step (a) further comprises receiving a password comprising the steps of:

(a1) displaying a screen location for entry of a password;

(a2) receiving characters of a password while not displaying any characters received;

(a3) displaying a clear password button;

(a4) disabling said clear password button until at least one character of a password has been received;

(a5) after at least one character has been received, enabling said clear password button; and

(a6) if all of said characters of a password are removed, disabling said clear password button.

7. The method of claim **5** wherein step (d) further comprises the steps of:

(d1) determining a type of operating system software contained in said target computer system; and

(d2) selecting a set of software for installation, from at least two sets of software on said storage device,

wherein said selected set of software matches said type of operating system software determined in step (d1).

8. The method of claim 5 wherein the checksum stored within the local computer system is further stored within the software executing within the local computer system.

9. A computer implemented method for remotely executing software from a local computer system to a remote computer system through a computer network, the method comprising the steps of:

(a) receiving data, on the local computer system, to select the remote computer system from one of a plurality of potential remote computer systems;

(b) displaying a screen location for entry of a password;

(c) receiving characters of a password while not displaying any characters received;

(d) displaying a clear password button;

(e) disabling said clear password button until at least one character of said password has been received;

(f) after at least one character of said password has been received, enabling said clear password button, enabling said clear password button, wherein said enabling provides a visual indication that at least one character has been received;

(g) if all of said characters of said password are removed, disabling said clear password button, wherein said disabling provides a visual indication that all characters have been removed;

(h) if said password received in steps (b) through (g) is correct, performing the following steps (i) through (k);

(i) connecting the local computer system to the remote computer system, wherein the local computer system operates as a remote terminal of the remote computer system;

(j) connecting a storage device on the local computer system as a storage device on the remote computer system, wherein the storage device contains the software to be executed on the remote computer system; and

(k) performing, on the remote computer system, the software stored on the storage device.

10. A computer implemented method for installing software from a local computer system to a target computer system through a computer network, the method comprising the steps of:

(a) receiving data to select the target computer system from one of a plurality of potential target computer systems;

(b) displaying a screen location for entry of a password;

(c) receiving characters of a password while not displaying any characters received;

(d) displaying a clear password button;

(e) disabling said clear password button until at least one character of said password has been received;

(f) after at least one character of said password has been received, enabling said clear password button, wherein said enabling provides a visual indication that at least one character has been received;

(g) if all of said characters of said password are removed, disabling said clear password button, wherein said disabling provides a visual indication that all characters have been removed;

(h) if said password received in steps (b) through (g) is correct, performing the following steps (i) through (k);

(i) connecting the local computer system to the target computer system, wherein the local computer system operates as a remote terminal of the target computer system;

(j) connecting a storage device, containing the software to be installed, on the local computer system as a storage device on the target computer system; and

(k) performing, on the target computer system, one or more installation programs stored on the storage device to perform the installation.

* * * * *